

版权注意事项：1、书籍版权归著者和出版社所有；  
2、本PDF仅用于个人获取知识，进行私底下知识交流；  
3、PDF获得者不得在互联网以任何目的进行传播；  
如有需要，请尽量购买正版实体书！支持书籍作者！！

以实战为导向，讲透Python网络爬虫各项核心技术和主流框架，帮助读者快速、深度掌握网络爬虫的爬取技术与反爬攻关技巧

韦玮  
◎  
著

```
subtract=subtract.copy()

subtract:

append(myset-{i})

subtract=newsubtract-{i}

extend(getsubset(myset-{i},newsubtract))

result

set(myset):

set(),myset] if myset else [myset]

extend(getsubset(myset,myset))

result

subset({'a','b','c','d'})

sorted(x) for x in result]

orted(toprint,key=lambda x:(len(x),x)):

subset(myset,subtract):

myset]<=1:

[]

subtract=subtract.copy()

subtract:

append(myset-{i}

subtract=newsubtract

extend(getsubset(myset,subtract))

result

set(myset):

set(),myset] if myset else [myset]

extend(getsubset(myset,myset))

result

subset({'a','b','c','d'})

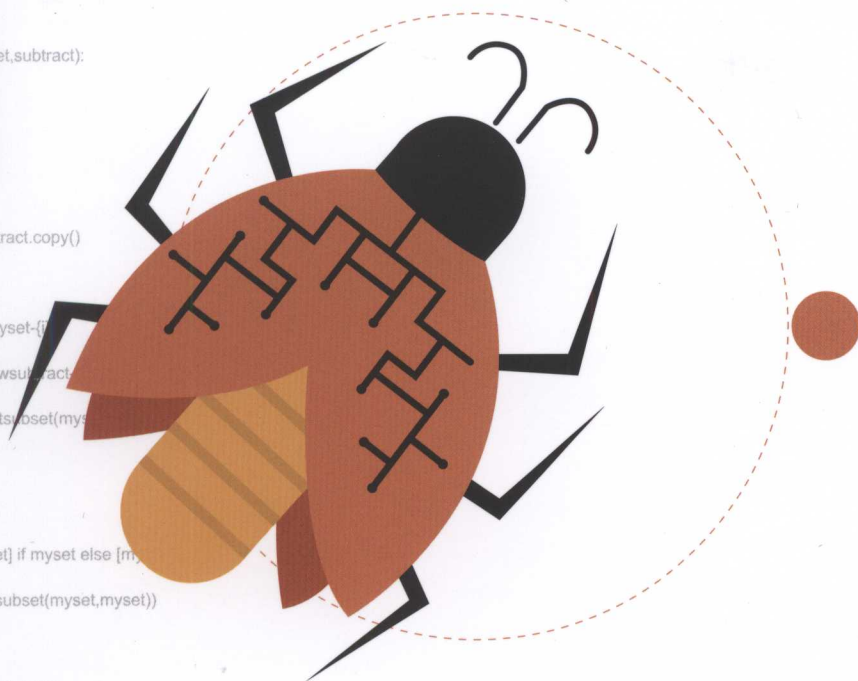
sorted(x) for x in result]

orted(toprint,key=lambda x:(len(x),x)):
```

Deep in Python Web Crawler  
Core Technology、Frame and Practices

# 精通Python 网络爬虫

核心技术、框架与项目实战



机械工业出版社  
China Machine Press



本书从技术、工具与实战3个维度讲解了Python网络爬虫：

**技术维度：**详细讲解了Python网络爬虫实现的核心技术，包括网络爬虫的工作原理、如何用urllib库编写网络爬虫、爬虫的异常处理、正则表达式、爬虫中Cookie的使用、爬虫的浏览器伪装技术、定向爬取技术、反爬虫技术，以及如何自己动手编写网络爬虫。

**工具维度：**以流行的Python网络爬虫框架Scrapy为对象，详细讲解了Scrapy的功能使用、高级技巧、架构设计、实现原理，以及如何通过Scrapy来更便捷、高效地编写网络爬虫。

**实战维度：**以实战为导向，是本书的主旨，除了完全通过手动编程实现网络爬虫和通过Scrapy框架实现网络爬虫的实战案例以外，本书还有博客爬取、图片爬取、模拟登录等多个综合性的网络爬虫实践案例。

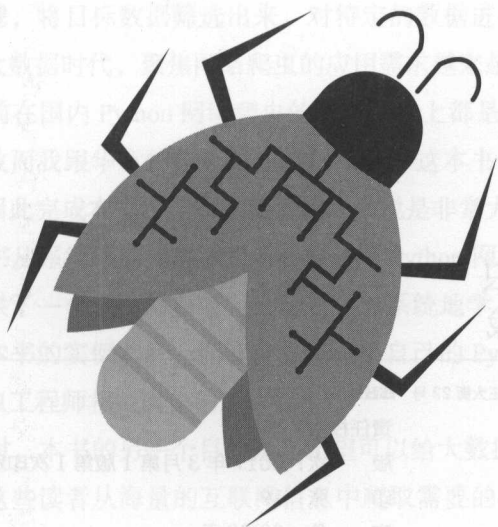
作者在Python领域有非常深厚的积累，不仅精通Python网络爬虫，在Python机器学习、Python数据分析与挖掘、Python Web开发等多个领域都有丰富的实战经验。

Deep in Python Web Crawler  
Core Technology, Frame and Practices

# 精通Python 网络爬虫

核心技术、框架与项目实战

韦玮◎著



机械工业出版社  
China Machine Press

## 图书在版编目 (CIP) 数据

精通 Python 网络爬虫：核心技术、框架与项目实战 / 韦玮著. —北京：机械工业出版社，2017.2

ISBN 978-7-111-56208-5

I. 精… II. 韦… III. 软件工具—程序设计 IV. TP311.561

中国版本图书馆 CIP 数据核字 (2017) 第 040626 号

## 精通 Python 网络爬虫 核心技术、框架与项目实战

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：何欣阳

责任校对：李秋荣

印 刷：中国电影出版社印刷厂

版 次：2017 年 3 月第 1 版第 1 次印刷

开 本：186mm×240mm 1/16

印 张：19

书 号：ISBN 978-7-111-56208-5

定 价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88379426 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

## Preface 前言

### 为什么写这本书

网络爬虫其实很早就出现了，最开始网络爬虫主要应用在各种搜索引擎中。在搜索引擎中，主要使用通用网络爬虫对网页进行爬取及存储。

随着大数据时代的到来，我们经常需要在海量数据的互联网环境中搜集一些特定的数据并对其进行分析，我们可以使用网络爬虫对这些特定的数据进行爬取，并对一些无关的数据进行过滤，将目标数据筛选出来。对特定的数据进行爬取的爬虫，我们将其称为聚焦网络爬虫。在大数据时代，聚焦网络爬虫的应用需求越来越大。

目前国内 Python 网络爬虫的书籍基本上都是从国外引进翻译的，国内的本版书籍屈指可数，故而我跟华章的副总编杨福川策划了这本书。本书的撰写过程中各方面的参考资料非常少，因此完成本书所花费的精力相对来说是非常大的。

本书从系统化的视角，为那些想学习 Python 网络爬虫或者正在研究 Python 网络爬虫的朋友们提供了一个全面的参考，让读者可以系统地学习 Python 网络爬虫的方方面面，在理解并掌握了本书的实例之后，能够独立编写出自己的 Python 网络爬虫项目，并且能够胜任 Python 网络爬虫工程师相关岗位的工作。

同时，本书的另一个目的是，希望可以给大数据或者数据挖掘方向的从业者一定的参考，以帮助这些读者从海量的互联网信息中爬取需要的数据。所谓巧妇难为无米之炊，有了这些数据之后，从事大数据或者数据挖掘方向工作的读者就可以进行后续的分析处理了。

### 本书的主要内容和特色

本书是一本系统介绍 Python 网络爬虫的书籍，全书注重实战，涵盖网络爬虫原理、如何手写 Python 网络爬虫、如何使用 Scrapy 框架编写网络爬虫项目等关于 Python 网络爬虫的方方面面。

本书的主要特色如下：

- 系统讲解 Python 网络爬虫的编写方法，体系清晰。
- 结合实战，让读者能够从零开始掌握网络爬虫的基本原理，学会编写 Python 网络爬虫以及 Scrapy 爬虫项目，从而编写出通用爬虫及聚焦爬虫，并掌握常见网站的爬虫反屏蔽手段。
- 有配套免费视频，对于书中的难点，读者可以直接观看作者录制的对应视频，加深理解。
- 拥有多个爬虫项目编写案例，比如博客类爬虫项目案例、图片类爬虫项目案例、模拟登录爬虫项目等。除此之外，还有很多不同种类的爬虫案例，可以让大家在理解这些案例之后学会各种类型爬虫的编写方法。

总之，在理解本书内容并掌握书中实例之后，读者将能胜任 Python 网络爬虫工程师方向的工作并学会各种类型网络爬虫项目的编写。此外，本书对于大数据或数据挖掘方向的从业者也非常有帮助，比如可以利用 Python 网络爬虫轻松获取所需的数据信息等。

## 本书面向的读者

- Python 网络爬虫初学者
- 网络爬虫工程师
- 大数据及数据挖掘工程师
- 高校计算机专业的学生
- 其他对 Python 或网络爬虫感兴趣的人员

## 如何阅读本书

本书分为四篇，共计 20 章。

第一篇为理论基础篇（第 1 ~ 2 章），主要介绍了网络爬虫的基础知识，让大家从零开始对网络爬虫有一个比较清晰的认识。

第二篇为核心技术篇（第 3 ~ 9 章），详细介绍了网络爬虫实现的核心技术，包括网络爬虫的工作原理、如何用 Urllib 库编写网络爬虫、爬虫的异常处理、正则表达式、爬虫中 Cookie 的使用、手写糗事百科爬虫、手写链接爬虫、手写微信爬虫、手写多线程爬虫、浏览器伪装技术、Python 网络爬虫的定向爬取技术及实例等。学完这一部分内容，读者就可以写出自己的爬虫了。这部分的爬虫编写采用的是一步步纯手写的方式进行的，没有采用框架。



第三篇为框架实现篇（第 10 ~ 17 章），主要详细介绍了如何用框架实现 Python 网络爬虫项目。使用框架实现 Python 网络爬虫项目相较于手写方式更加便捷，主要包括 Python 爬虫框架分类、Scrapy 框架在各系统中的安装以及如何避免各种“坑”、如何用 Scrapy 框架编写爬虫项目、Scrapy 框架架构详解、Scrapy 的中文输出与存储、在 Scrapy 中如何使用 for 循环实现自动网页爬虫、如何通过 CrawlSpider 实现自动网页爬虫、如何将爬取的内容写进数据库等。其中第 12 章为基础部分，读者需要着重掌握。

第四篇为项目实战篇（第 18 ~ 20 章），分别讲述了博客类爬虫项目、图片类爬虫项目、模拟登录爬虫项目的编程及实现。其中，也会涉及验证码处理等方面的难点知识，帮助读者通过实际的项目掌握网络爬虫项目的编写。

## 勘误和支持

由于作者的水平有限，书中难免有一些错误或不准确的地方，恳请各位读者不吝指正。

相关建议各位可以通过微博 @韦玮 pig 或通过 QQ 公众号 a67899 或微信公众平台 weijc7789（可以直接扫描下方二维码添加）进行反馈，也可以直接向邮箱 [ceo@iqianyue.com](mailto:ceo@iqianyue.com) 发送邮件，期待能够收到各位读者的意见和建议，欢迎来信。



扫描关注 QQ 公众号



扫描关注微信公众号

## 致谢

感谢机械工业出版社华章公司的副总编杨福川老师与编辑李艺老师，在近一年的时间里，是你们一次次在我遇到困难的时候，给予我鼓励，让我可以坚持写下去。创作一本图书是非常艰苦的，除了技术知识等因素之外，还需要非常大的毅力。特别感谢杨福川在写作过程中对我各方面的支持，尤其是对我毅力的培养。

感谢 CSDN、51CTO 与极客学院，因为你们，让我在这个领域获得了更多的学员与支持。

感谢恩师何云景教授对我创业方面的帮助，因为有您，我才拥有了一个更好的创业开端及工作环境。



## 特别致谢

最后，需要特别感谢的是我的女友，因为编写这本书，少了很多陪你的时间，感谢你的不离不弃与理解包容。希望未来可以加倍弥补你那些错过吃的美食和那些错过逛的街道。

同时，也要感谢你帮我完成书稿的校对工作，谢谢你的付出与支持。因为有了你默默的付出，我才能坚定地走下去；因为有了你不断的支持，我才可以安心地往前冲。

感谢爷爷从小对我人生观、价值观的培养，您是一个非常有思想的人。

感谢远方的父母、叔叔、姐姐，那些亲情的陪伴是我最珍贵的财富。

谨以此书献给热爱 Python 的朋友们!

## Contents 目 录

### 前 言

## 第一篇 理论基础篇

### 第1章 什么是网络爬虫 ..... 3

#### 1.1 初识网络爬虫 ..... 3

#### 1.2 为什么要学网络爬虫 ..... 4

#### 1.3 网络爬虫的组成 ..... 5

#### 1.4 网络爬虫的类型 ..... 6

#### 1.5 爬虫扩展——聚焦爬虫 ..... 7

#### 1.6 小结 ..... 8

### 第2章 网络爬虫技能总览 ..... 9

#### 2.1 网络爬虫技能总览图 ..... 9

#### 2.2 搜索引擎核心 ..... 10

#### 2.3 用户爬虫的那些事儿 ..... 11

#### 2.4 小结 ..... 12

## 第二篇 核心技术篇

### 第3章 网络爬虫实现原理与实现技术 ..... 15

#### 3.1 网络爬虫实现原理详解 ..... 15

#### 3.2 爬行策略 ..... 17

#### 3.3 网页更新策略 ..... 18

#### 3.4 网页分析算法 ..... 20

#### 3.5 身份识别 ..... 21

#### 3.6 网络爬虫实现技术 ..... 21

#### 3.7 实例——metaseeker ..... 22

#### 3.8 小结 ..... 27

### 第4章 Urllib 库与 URLError 异常处理 ..... 29

#### 4.1 什么是 Urllib 库 ..... 29

#### 4.2 快速使用 Urllib 爬取网页 ..... 30

#### 4.3 浏览器的模拟——Headers 属性 ..... 34

#### 4.4 超时设置 ..... 37

#### 4.5 HTTP 协议请求实战 ..... 39

#### 4.6 代理服务器的设置 ..... 44

#### 4.7 DebugLog 实战 ..... 45

#### 4.8 异常处理神器——URLError 实战 ..... 46

#### 4.9 小结 ..... 51

### 第5章 正则表达式与 Cookie 的使用 ..... 52

#### 5.1 什么是正则表达式 ..... 52

5.2 正则表达式基础知识 .....	52	8.3 爬虫的浏览器伪装技术 实战 .....	117
5.3 正则表达式常见函数 .....	61	8.4 小结 .....	121
5.4 常见实例解析 .....	64	<b>第9章 爬虫的定向爬取技术</b> .....	122
5.5 什么是 Cookie .....	66	9.1 什么是爬虫的定向爬取技术 .....	122
5.6 Cookiejar 实战精析 .....	66	9.2 定向爬取的相关步骤与策略 .....	123
5.7 小结 .....	71	9.3 定向爬取实战 .....	124
<b>第6章 手写 Python 爬虫</b> .....	73	9.4 小结 .....	130
6.1 图片爬虫实战 .....	73	<b>第三篇 框架实现篇</b>	
6.2 链接爬虫实战 .....	78	<b>第10章 了解 Python 爬虫框架</b> .....	133
6.3 糗事百科爬虫实战 .....	80	10.1 什么是 Python 爬虫框架 .....	133
6.4 微信爬虫实战 .....	82	10.2 常见的 Python 爬虫框架 .....	133
6.5 什么是多线程爬虫 .....	89	10.3 认识 Scrapy 框架 .....	134
6.6 多线程爬虫实战 .....	90	10.4 认识 Crawley 框架 .....	135
6.7 小结 .....	98	10.5 认识 Portia 框架 .....	136
<b>第7章 学会使用 Fiddler</b> .....	99	10.6 认识 newspaper 框架 .....	138
7.1 什么是 Fiddler .....	99	10.7 认识 Python-goose 框架 .....	139
7.2 爬虫与 Fiddler 的关系 .....	100	10.8 小结 .....	140
7.3 Fiddler 的基本原理与基本 界面 .....	100	<b>第11章 爬虫利器——Scrapy 安装与配置</b> .....	141
7.4 Fiddler 捕获会话功能 .....	102	11.1 在 Windows7 下安装及配置 Scrapy 实战详解 .....	141
7.5 使用 QuickExec 命令行 .....	104	11.2 在 Linux (Centos) 下安装及配置 Scrapy 实战详解 .....	147
7.6 Fiddler 断点功能 .....	106	11.3 在 MAC 下安装及配置 Scrapy 实战详解 .....	158
7.7 Fiddler 会话查找功能 .....	111	11.4 小结 .....	161
7.8 Fiddler 的其他功能 .....	111		
7.9 小结 .....	113		
<b>第8章 爬虫的浏览器伪装技术</b> .....	114		
8.1 什么是浏览器伪装技术 .....	114		
8.2 浏览器伪装技术准备工作 .....	115		

## 第12章 开启Scrapy爬虫项目之旅..... 162

- 12.1 认识Scrapy项目的目录结构..... 162
- 12.2 用Scrapy进行爬虫项目管理..... 163
- 12.3 常用工具命令..... 166
- 12.4 实战: Items的编写..... 181
- 12.5 实战: Spider的编写..... 183
- 12.6 XPath基础..... 187
- 12.7 Spider类参数传递..... 188
- 12.8 用XMLFeedSpider来分析XML源..... 191
- 12.9 学会使用CSVFeedSpider..... 197
- 12.10 Scrapy爬虫多开技能..... 200
- 12.11 避免被禁止..... 206
- 12.12 小结..... 212

## 第13章 Scrapy核心架构..... 214

- 13.1 初识Scrapy架构..... 214
- 13.2 常用的Scrapy组件详解..... 215
- 13.3 Scrapy工作流..... 217
- 13.4 小结..... 219

## 第14章 Scrapy中文输出与存储..... 220

- 14.1 Scrapy的中文输出..... 220
- 14.2 Scrapy的中文存储..... 223
- 14.3 输出中文到JSON文件..... 225
- 14.4 小结..... 230

## 第15章 编写自动爬取

### 网页的爬虫..... 231

- 15.1 实战: items的编写..... 231

- 15.2 实战: pipelines的编写..... 233

- 15.3 实战: settings的编写..... 234

- 15.4 自动爬虫编写实战..... 234

- 15.5 调试与运行..... 239

- 15.6 小结..... 242

## 第16章 CrawlSpider..... 243

- 16.1 初识CrawlSpider..... 243

- 16.2 链接提取器..... 244

- 16.3 实战: CrawlSpider实例..... 245

- 16.4 小结..... 249

## 第17章 Scrapy高级应用..... 250

- 17.1 如何在Python3中操作数据库..... 250

- 17.2 爬取内容写进MySQL..... 254

- 17.3 小结..... 259

## 第四篇 项目实战篇

## 第18章 博客类爬虫项目..... 263

- 18.1 博客类爬虫项目功能分析..... 263

- 18.2 博客类爬虫项目实现思路..... 264

- 18.3 博客类爬虫项目编写实战..... 264

- 18.4 调试与运行..... 274

- 18.5 小结..... 275

## 第19章 图片类爬虫项目..... 276

- 19.1 图片类爬虫项目功能分析..... 276

19.2 图片类爬虫项目实现	277
思路	277
19.3 图片类爬虫项目	277
编写实战	277
19.4 调试与运行	281
19.5 小结	282

## 第 20 章 模拟登录爬虫项目 ..... 283

20.1 模拟登录爬虫项目功能分析	283
20.2 模拟登录爬虫项目实现思路	283
20.3 模拟登录爬虫项目编写实战	284
20.4 调试与运行	292
20.5 小结	294

## 第一篇 Part 1

# 理论基础篇

随着互联网时代的到来,网络爬虫在互联网中的地位越来越重要。互联网中的数据是海量的,如何自动高效地获取互联网中我们感兴趣的信息并为我们所用是一个重要的问题,而爬虫技术就是为了解决这些问题而生的。我们感兴趣的信息分为不同的类型:如果只是做搜索引擎,那么感兴趣的信息就是互联网中尽可能多的高质量网页;如果要获取某一垂直领域的数据或者有明确的检索需求,那么感兴趣的信息就是根据我们的检索和需求所定位的这些信息,此时,需要过滤掉一些无用信息。前者我们称为通用网络爬虫,后者我们称为垂直网络爬虫。

## 1.1 初识网络爬虫

网络爬虫又称网络蜘蛛、网络蚂蚁、网络机器人等,可以自动化地浏览网络中的信息,当需要浏览信息的时候需要按照我们制定的规则进行,这些规则我们称之为网络爬虫算法。使用Python可以很方便地编写网络爬虫程序,实现网络信息的自动化检索。

- 第1章 什么是网络爬虫
- 第2章 网络爬虫技能总览

搜索引擎离不开网络爬虫。网络爬虫又叫作百度蜘蛛(Baiduspider)。百度蜘蛛每天会在海量的互联网页面中进行抓取,抓取网页内容并收录。当用户在百度搜索引擎上搜索特定关键词时,百度将对关键词进行分析处理,从收录的网页中找出相关网页,按照一定的排名规则进行排序并将结果呈现给用户。在这个过程中,百度蜘蛛起到了至关重要的作用。那么,如何抓取互联网中更多的优质网页?又如何筛选这些收录的页面?这些都是由百度蜘蛛爬虫的算法决定的。采用不同的算法,爬虫的运行效率会不同,爬取结果也会有所差异。所以,我们在研究爬虫的时候,不仅要了解爬虫如何实现,还要知道一些常见爬虫的



网络爬虫也叫做网络机器人，可以代替人们自动地在互联网中进行数据信息的采集与整理。在大数据时代，信息的采集是一项重要的工作，如果单纯靠人力进行信息采集，不仅低效繁琐，搜集的成本也会提高。此时，我们可以使用网络爬虫对数据信息进行自动采集，比如应用于搜索引擎中对站点进行爬取收录，应用于数据分析与挖掘中对数据进行采集，应用于金融分析中对金融数据进行采集，除此之外，还可以将网络爬虫应用于舆情监测与分析、目标客户数据的收集等各个领域。当然，要学习网络爬虫开发，首先需要认识网络爬虫，在本篇中，我们将带领大家一起认识几种典型的网络爬虫，并了解网络爬虫的各项常见功能。

## 第1章 Chapter 1

## 什么是网络爬虫

随着大数据时代的来临，网络爬虫在互联网中的地位将越来越重要。互联网中的数据是海量的，如何自动高效地获取互联网中我们感兴趣的信息并为我们所用是一个重要的问题，而爬虫技术就是为了解决这些问题而生的。我们感兴趣的信息分为不同的类型：如果只是做搜索引擎，那么感兴趣的信息就是互联网中尽可能多的高质量网页；如果要获取某一垂直领域的数据或者有明确的检索需求，那么感兴趣的信息就是根据我们的检索和需求所定位的这些信息，此时，需要过滤掉一些无用信息。前者我们称为通用网络爬虫，后者我们称为聚焦网络爬虫。

## 1.1 初识网络爬虫

网络爬虫又称网络蜘蛛、网络蚂蚁、网络机器人等，可以自动化浏览网络中的信息，当然浏览信息的时候需要按照我们制定的规则进行，这些规则我们称之为网络爬虫算法。使用Python可以很方便地编写出爬虫程序，进行互联网信息的自动化检索。

搜索引擎离不开爬虫，比如百度搜索引擎的爬虫叫作百度蜘蛛（Baiduspider）。百度蜘蛛每天会在海量的互联网信息中进行爬取，爬取优质信息并收录，当用户在百度搜索引擎上检索对应关键词时，百度将对关键词进行分析处理，从收录的网页中找出相关网页，按照一定的排名规则进行排序并将结果展现给用户。在这个过程中，百度蜘蛛起到了至关重要的作用。那么，如何覆盖互联网中更多的优质网页？又如何筛选这些重复的页面？这些都是由百度蜘蛛爬虫的算法决定的。采用不同的算法，爬虫的运行效率会不同，爬取结果也会有所差异。所以，我们在研究爬虫的时候，不仅要了解爬虫如何实现，还需要知道一些常见爬虫的

算法,如果有必要,我们还需要自己去制定相应的算法,这些在后面都会为大家详细地讲解,在此,我们仅需要对爬虫的概念有一个基本的了解。

除了百度搜索引擎离不开爬虫以外,其他搜索引擎也离不开爬虫,它们也拥有自己的爬虫。比如 360 的爬虫叫 360Spider,搜狗的爬虫叫 Sogouspider,必应的爬虫叫 Bingbot。

如果想自己实现一款小型的搜索引擎,我们也可以编写出自己的爬虫去实现,当然,虽然可能在性能或者算法上比不上主流的搜索引擎,但是个性化的程度会非常高,并且也有利于我们更深层地理解搜索引擎内部的工作原理。

大数据时代也离不开爬虫,比如在进行大数据分析或数据挖掘时,我们可以去一些比较大型的官方站点下载数据源。但这些数据源比较有限,那么如何才能获取更多更高质量的数据源呢?此时,我们可以编写自己的爬虫程序,从互联网中进行数据信息的获取。所以在未来,爬虫的地位会越来越重要。

## 1.2 为什么要学网络爬虫

在上一节中,我们初步认识了网络爬虫,但是为什么要学习网络爬虫呢?要知道,只有清晰地知道我们的学习目的,才能够更好地学习这一项知识,所以在这一节中,我们将会为大家分析一下学习网络爬虫的原因。

当然,不同的人学习爬虫,可能目的有所不同,在此,我们总结了 4 种常见的学习爬虫的原因。

1) 学习爬虫,可以私人订制一个搜索引擎,并且可以对搜索引擎的数据采集工作原理进行更深层地理解。

有的朋友希望能够深层次地了解搜索引擎的爬虫工作原理,或者希望自己能够开发出一款私人搜索引擎,那么此时,学习爬虫是非常有必要的。简单来说,我们学会了爬虫编写之后,就可以利用爬虫自动地采集互联网中的信息,采集回来后进行相应的存储或处理,在需要检索某些信息的时候,只需在采集回来的信息中进行检索,即实现了私人的搜索引擎。当然,信息怎么爬取、怎么存储、怎么进行分词、怎么进行相关性计算等,都是需要进行设计的,爬虫技术主要解决信息爬取的问题。

2) 大数据时代,要进行数据分析,首先要有数据源,而学习爬虫,可以让我们获取更多的数据源,并且这些数据源可以按我们的目的进行采集,去掉很多无关数据。

在进行大数据分析或者进行数据挖掘的时候,数据源可以从某些提供数据统计的网站获得,也可以从某些文献或内部资料中获得,但是这些获得数据的方式,有时很难满足我们对数据的需求,而手动从互联网中去寻找这些数据,则耗费的精力过大。此时就可以利用爬虫技术,自动地从互联网中获取我们感兴趣的数据内容,并将这些数据内容爬取回来,作为我们的数据源,从而进行更深层的数据分析,并获得更多有价值的信息。

3) 对于很多 SEO 从业者来说,学习爬虫,可以更深层地理解搜索引擎爬虫的工作原

理，从而可以更好地进行搜索引擎优化。

既然是搜索引擎优化，那么就必须要对搜索引擎的工作原理非常清楚，同时也需要掌握搜索引擎爬虫的工作原理，这样在进行搜索引擎优化时，才能知己知彼，百战不殆。

4) 从就业的角度来说，爬虫工程师目前来说属于紧缺人才，并且薪资待遇普遍较高，所以，深层次地掌握这门技术，对于就业来说，是非常有利的。

有些朋友学习爬虫可能为了就业或者跳槽。从这个角度来说，爬虫工程师方向是不错的选择之一，因为目前爬虫工程师的需求越来越大，而能够胜任这方面岗位的人员较少，所以属于一个比较紧缺的职业方向，并且随着大数据时代的来临，爬虫技术的应用将越来越广泛，在未来会拥有很好的发展空间。

除了以上为大家总结的4种常见的学习爬虫的原因外，可能你还有一些其他学习爬虫的原因，总之，不管是什么原因，理清自己学习的目的，就可以更好地去研究一门知识技术，并坚持下来。

### 1.3 网络爬虫的组成

接下来，我们将介绍网络爬虫的组成。网络爬虫由控制节点、爬虫节点、资源库构成。

图 1-1 所示是网络爬虫的控制节点和爬虫节点的结构关系。

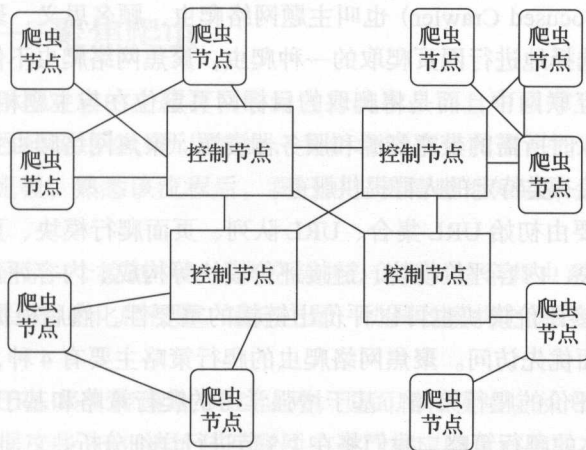


图 1-1 网络爬虫的控制节点和爬虫节点的结构关系

可以看到，网络爬虫中可以有多个控制节点，每个控制节点下可以有多个爬虫节点，控制节点之间可以互相通信，同时，控制节点和其下的各爬虫节点之间也可以进行互相通信，属于同一个控制节点下的各爬虫节点间，亦可以互相通信。

控制节点，也叫作爬虫的中央控制器，主要负责根据 URL 地址分配线程，并调用爬虫



节点进行具体的爬行。

爬虫节点会按照相关的算法,对网页进行具体的爬行,主要包括下载网页以及对网页的文本进行处理,爬行后,会将对应的爬行结果存储到对应的资源库中。

## 1.4 网络爬虫的类型

现在我们已经基本了解了网络爬虫的组成,那么网络爬虫具体有哪些类型呢?

网络爬虫按照实现的技术和结构可以分为通用网络爬虫、聚焦网络爬虫、增量式网络爬虫、深层网络爬虫等类型。在实际的网络爬虫中,通常是这几类爬虫的组合物。

首先我们为大家介绍**通用网络爬虫**(General Purpose Web Crawler)。通用网络爬虫又叫作**全网爬虫**,顾名思义,通用网络爬虫爬取的目标资源在全互联网中。通用网络爬虫所爬取的目标数据是巨大的,并且爬行的范围也是非常大的,正是由于其爬取的数据是海量数据,故而对于这类爬虫来说,其爬取的性能要求是非常高的。这种网络爬虫主要应用于大型搜索引擎中,有非常高的应用价值。

通用网络爬虫主要由初始 URL 集合、URL 队列、页面爬行模块、页面分析模块、页面数据库、链接过滤模块等构成。通用网络爬虫在爬行的时候会采取一定的爬行策略,主要有深度优先爬行策略和广度优先爬行策略。具体的爬行策略,我们将在第 3 章讲解,在此,我们只需要知道通用网络爬虫的基本构成和主要的爬行策略。

**聚焦网络爬虫**(Focused Crawler)也叫**主题网络爬虫**,顾名思义,聚焦网络爬虫是按照预先定义好的主题有选择地进行网页爬取的一种爬虫,聚焦网络爬虫不像通用网络爬虫一样将目标资源定位在全互联网中,而是将爬取的目标网页定位在与主题相关的页面中,此时,可以大大节省爬虫爬取时所需的带宽资源和服务器资源。聚焦网络爬虫主要应用在对特定信息的爬取中,主要为某一类特定的人群提供服务。

聚焦网络爬虫主要由初始 URL 集合、URL 队列、页面爬行模块、页面分析模块、页面数据库、链接过滤模块、内容评价模块、链接评价模块等构成。内容评价模块可以评价内容的重要性,同理,链接评价模块也可以评价出链接的重要性,然后根据链接和内容的重要性,可以确定哪些页面优先访问。聚焦网络爬虫的爬行策略主要有 4 种,即基于内容评价的爬行策略、基于链接评价的爬行策略、基于增强学习的爬行策略和基于语境图的爬行策略。关于聚焦网络爬虫具体的爬行策略,我们将在 1.5 节进行详细分析。

**增量式网络爬虫**(Incremental Web Crawler),所谓增量式,对应着增量式更新。增量式更新指的是在更新的时候只更新改变的地方,而未改变的地方则不更新,所以增量式网络爬虫,在爬取网页的时候,只爬取内容发生变化的网页或者新产生的网页,对于未发生内容变化的网页,则不会爬取。增量式网络爬虫在一定程度上能够保证所爬取的页面,尽可能是新页面。

**深层网络爬虫**(Deep Web Crawler),可以爬取互联网中的深层页面,在此我们首先需要

了解深层页面的概念。

在互联网中,网页按存在方式分类,可以分为表层页面和深层页面。所谓的表层页面,指的是不需要提交表单,使用静态的链接就能够到达的静态页面;而深层页面则隐藏在表单后面,不能通过静态链接直接获取,是需要提交一定的关键词之后才能够获取到的页面。在互联网中,深层页面的数量往往比表层页面的数量要多很多,故而,我们需要想办法爬取深层页面。

爬取深层页面,需要想办法自动填写好对应表单,所以,深层网络爬虫最重要的部分即为表单填写部分。

深层网络爬虫主要由 URL 列表、LVS 列表(LVS 指的是标签/数值集合,即填充表单的数据源)、爬行控制器、解析器、LVS 控制器、表单分析器、表单处理器、响应分析器等部分构成。

深层网络爬虫表单的填写有两种类型:第一种是基于领域知识的表单填写,简单来说就是建立一个填写表单的关键词库,在需要填写的时候,根据语义分析选择对应的关键词进行填写;第二种是基于网页结构分析的表单填写,简单来说,这种填写方式一般是领域知识有限的情况下使用,这种方式会根据网页结构进行分析,并自动地进行表单填写。

以上,为大家介绍了网络爬虫中常见的几种类型,希望读者能够对网络爬虫的分类有一个基本的了解。

## 1.5 爬虫扩展——聚焦爬虫

由于聚焦爬虫可以按对应的主题有目的地进行爬取,并且可以节省大量的服务器资源和带宽资源,具有很强的实用性,所以在此,我们将对聚焦爬虫进行详细讲解。图 1-2 所示为聚焦爬虫运行的流程,熟悉该流程后,我们可以更清晰地知道聚焦爬虫的工作原理和过程。

首先,聚焦爬虫拥有一个控制中心,该控制中心负责对整个爬虫系统进行管理和监控,主要包括控制用户交互、初始化爬行器、确定主题、协调各模块之间的工作、控制爬行过程等方面。

然后,将初始的 URL 集合传递给 URL 队列,页面爬行模块会从 URL 队列中读取第一批 URL 列表,然后根据这些 URL 地址从互联网中进行相应的页面爬取。爬取后,将爬取到的内容传到页面数据库中存储,同时,在爬行过程中,会爬取到一些新的 URL,此时,需要根据我们所定的主题使用链接过滤模块过滤掉无关链接,再将剩下的 URL 链接根据主题使用链接评价模块或内容评价模块进行优先级的排序。完成后,将新的 URL 地址传递到 URL 队列中,供页面爬行模块使用。另一方面,将页面爬取并存放到页面数据库后,需要根据主题使用页面分析模块对爬取到的页面进行页面分析处理,并根据处理结果建立索引数据库,用户检索对应信息时,可以从索引数据库中进行相应的检索,并得到对应的结果。



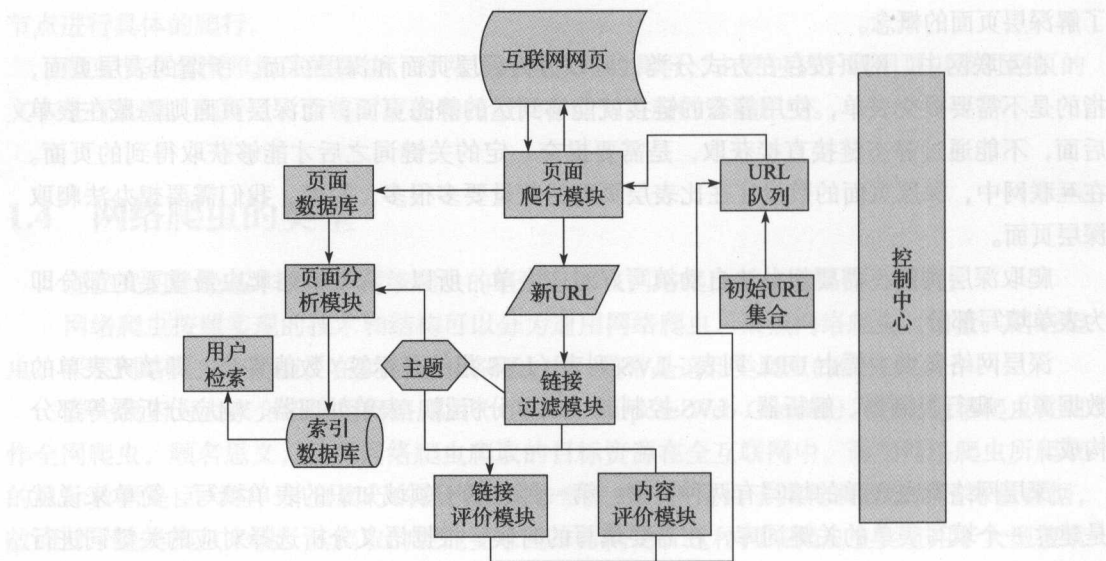


图 1-2 聚焦爬虫运行的流程

这就是聚焦爬虫的主要工作流程，了解聚焦爬虫的主要工作流程有助于我们编写聚焦爬虫，使编写的思路更加清晰。

## 1.6 小结

1) 网络爬虫也叫作网络蜘蛛、网络蚂蚁、网络机器人等，可以自动地浏览网络中的信息，当然浏览信息的时候需要按照我们制定的规则去浏览，这些规则我们将其称为网络爬虫算法。使用 Python 可以很方便地编写出爬虫程序，进行互联网信息的自动化检索。

2) 学习爬虫，可以：①私人订制一个搜索引擎，并且可以对搜索引擎的数据采集工作原理，进行更深层次地理解；②为大数据分析提供更多高质量的数据源；③更好地研究搜索引擎优化；④解决就业或跳槽的问题。

3) 网络爬虫由控制节点、爬虫节点、资源库构成。

4) 网络爬虫按照实现的技术和结构可以分为通用网络爬虫、聚焦网络爬虫、增量式网络爬虫、深层网络爬虫等类型。在实际的网络爬虫中，通常是这几类爬虫的组合物。

5) 聚焦网络爬虫主要由初始 URL 集合、URL 队列、页面爬行模块、页面分析模块、页面数据库、链接过滤模块、内容评价模块、链接评价模块等构成。

## 网络爬虫技能总览

在上一章中，我们已经初步认识了网络爬虫，那么网络爬虫具体能做什么呢？用网络爬虫又能做哪些有趣的事呢？在本章中我们将为大家具体讲解。

### 2.1 网络爬虫技能总览图

如图 2-1 所示，我们总结了网络爬虫的常用功能。

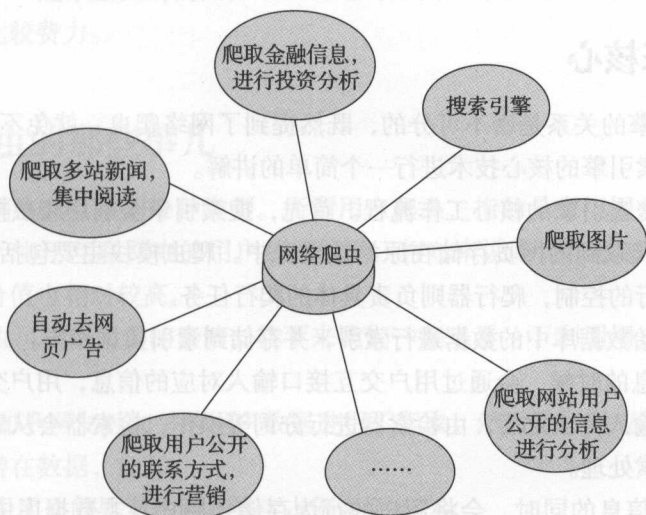


图 2-1 网络爬虫技能示意图

在图 2-1 中可以看到,网络爬虫可以代替手工做很多事情,比如可以用于做搜索引擎,也可以爬取网站上面的图片,比如有些朋友将某些网站上的图片全部爬取下来,集中进行浏览,同时,网络爬虫也可以用于金融投资领域,比如可以自动爬取一些金融信息,并进行投资分析等。

有时,我们比较喜欢的新闻网站可能有几个,每次都要分别打开这些新闻网站进行浏览,比较麻烦。此时可以利用网络爬虫,将这多个新闻网站中的新闻信息爬取下来,集中进行阅读。

有时,我们在浏览网页上的信息的时候,会发现有很多广告。此时同样可以利用爬虫将对应网页上的信息爬取过来,这样就可以自动的过滤掉这些广告,方便对信息的阅读与使用。

有时,我们需要进行营销,那么如何找到目标客户以及目标客户的联系方式是一个关键问题。我们可以手动地在互联网中寻找,但是这样的效率会很低。此时,我们利用爬虫,可以设置对应的规则,自动地从互联网中采集目标用户的联系方式等数据,供我们进行营销使用。

有时,我们想对某个网站的用户信息进行分析,比如分析该网站的用户活跃度、发言数、热门文章等信息,如果我们不是网站管理员,手工统计将是一个非常庞大的工程。此时,可以利用爬虫轻松将这些数据采集到,以便进行进一步分析,而这一切爬取的操作,都是自动进行的,我们只需要编写好对应的爬虫,并设计好对应的规则即可。

除此之外,爬虫还可以实现很多强大的功能。总之,爬虫的出现,可以在一定程度上代替手工访问网页,从而,原先我们需要人工去访问互联网信息的操作,现在都可以用爬虫自动化实现,这样可以更高效率地利用好互联网中的有效信息。

## 2.2 搜索引擎核心

爬虫与搜索引擎的关系是密不可分的,既然提到了网络爬虫,就免不了提到搜索引擎,在此,我们将对搜索引擎的核心技术进行一个简单的讲解。

图 2-2 所示为搜索引擎的核心工作流程。首先,搜索引擎会利用爬虫模块去爬取互联网中的网页,然后将爬取到的网页存储在原始数据库中。爬虫模块主要包括控制器和爬行器,控制器主要进行爬行的控制,爬行器则负责具体的爬行任务。

然后,会对原始数据库中的数据进行索引,并存储到索引数据库中。

当用户检索信息的时候,会通过用户交互接口输入对应的信息,用户交互接口相当于搜索引擎的输入框,输入完成之后,由检索器进行分词等操作,检索器会从索引数据库中获取数据进行相应的检索处理。

用户输入对应信息的同时,会将用户的行为存储到用户日志数据库中,比如用户的 IP 地址、用户所输入的关键词等等。随后,用户日志数据库中的数据会交由日志分析器进行处

理。日志分析器会根据大量的用户数据去调整原始数据库和索引数据库,改变排名结果或进行其他操作。

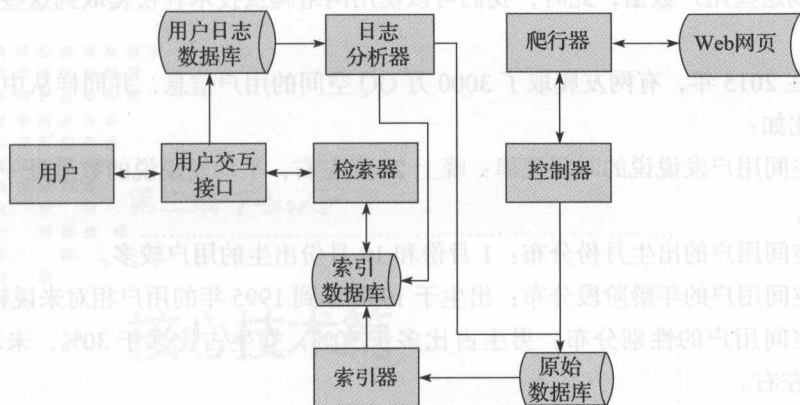


图 2-2 搜索引擎的核心工作流程

以上就是搜索引擎核心工作流程的简要概述,可能大家对索引和检索的概念还不太能区分,在此我为大家详细讲一下。

简单来说,检索是一种行为,而索引是一种属性。比如一家超市,里面有大量的商品,为了能够快速找到这些商品,我们会将这些商品进行分组,比如有日常用品类商品、饮料类商品、服装类商品等组别,此时,这些商品的组名我们称之为索引,索引由索引器控制。如果,有一个用户想要找到某一个商品,那么需要在超市的大量商品中寻找,这个过程,我们称之为检索。如果有一个好的索引,则可以提高检索的效率;若没有索引,则检索的效率会很低。比如,一个超市里面的商品如果没有进行分类,那么用户要在海量的商品中寻找某一种商品,则会比较费力。

## 2.3 用户爬虫的那些事儿

用户爬虫是网络爬虫中的一种类型。所谓用户爬虫,指的是专门用来爬取互联网中用户数据的一种爬虫。由于互联网中的用户数据信息,相对来说是比较敏感的数据信息,所以,用户爬虫的利用价值也相对较高。

利用用户爬虫可以做大量的事情,接下来我们一起来看一下利用用户爬虫所做的一些有趣的事情吧。

2015年,有知乎网友对知乎的用户数据进行了爬取,然后进行对应的数据分析,便得到了知乎上大量的潜在数据,比如:

- 知乎上注册用户的男女比例:男生占例多于60%。
- 知乎上注册用户的地区:北京的人口占据比重最大,多于30%。



□ 知乎上注册用户从事的行业：从事互联网行业的用户占据比重最大，同样多于 30%。

除此之外，只要我们细心发掘，还可以挖掘出更多的潜在数据，而要分析这些数据，则必须要获取到这些用户数据，此时，我们可以使用网络爬虫技术轻松爬取到这些有用的用户信息。

同样，在 2015 年，有网友爬取了 3000 万 QQ 空间的用户信息，并同样从中获得了大量潜在数据，比如：

- QQ 空间用户发说说的时间规律：晚上 22 点左右，平均发说说的数量是一天中最多的时候。
- QQ 空间用户的出生月份分布：1 月份和 10 月份出生的用户较多。
- QQ 空间用户的年龄阶段分布：出生于 1990 年到 1995 年的用户相对来说较多。
- QQ 空间用户的性别分布：男生占比多于 50%，女生占比多于 30%，未填性别的占 10% 左右。

除了以上两个例子之外，用户爬虫还可以做很多事情，比如爬取淘宝的用户信息，可以分析淘宝用户喜欢什么商品，从而更有利于我们对商品的定位等。

由此可见，利用用户爬虫可以获得很多有趣的潜在信息，那么这些爬虫难吗？其实不难，在阅读完本书后，相信你也能写出这样的爬虫。

## 2.4 小结

1) 爬虫的出现，可以在一定程度上代替手工访问网页，所以，原先我们需要人工去访问互联网信息的操作，现在都可以用爬虫自动化实现，这样可以更高效地利用好互联网中的有效信息。

2) 检索是一种行为，而索引是一种属性。如果有一个好的索引，则可以提高检索的效率，若没有索引，则检索的效率会很低。

3) 用户爬虫是网络爬虫的其中一种类型。所谓用户爬虫，即专门用来爬取互联网中用户数据的一种爬虫。由于互联网中的用户数据信息，相对来说是比较敏感的数据信息，所以，用户爬虫的利用价值也相对较高。

## 第二篇 *Part 2*

# 核心技术篇

我们已经初步认识了网络爬虫，并了解了网络爬虫的应用领域。在这一章中，我们将学习网络爬虫的实现原理及其实现技术，并使用 `newscrawler` 为大家做一个简单的爬虫案例。

## 3.1 网络爬虫实现原理详解

不同类型的网络爬虫，其实现原理也是不同的，但在这类实现原理中，会存在很多共性。在此，我们将以两种典型的网络爬虫为例（即通用网络爬虫和搜索引擎爬虫），分别为大家讲解网络爬虫的实现原理。

### 1. 通用网络爬虫

- 第3章 网络爬虫实现原理与实现技术
- 第4章 `Urllib` 库与 `URLError` 异常处理
- 第5章 正则表达式与 `Cookie` 的使用
- 第6章 手写 Python 爬虫
- 第7章 学会使用 `Fiddler`
- 第8章 爬虫的浏览器伪装技术
- 第9章 爬虫的定向爬取技术

首先我们来看通用网络爬虫的实现原理及过程，如图 3-1 所示。

1) 获取初始的 URL。初始的 URL 可以由用户人为地指定，也可以由用户指定的某个或多个初始爬取 URL。

2) 根据初始的 URL 爬取网页。获取了初始的 URL 地址之后，首先需要爬取对应 URL 地址中的网页，爬取了网页的 URL 地址中的网页后，将网页存储到原始数据库中，并且在爬取网页的同时，发现新的 URL 地址，同时将这些已爬取的 URL 地址存储到一个 URL 列表中，用于去重及判断爬取的进程。

3) 将新的 URL 放入 URL 队列中。在第 2 步中，获取了下一个新的 URL 地址之后，会将新的 URL 地址放入 URL 队列中。

4) 从 URL 队列中读取新的 URL，并依据新的 URL 爬取网页，同时从新网页中获取新



口知乎上注册用户从事的行业：从事互联网行业的用户占据比重最大，同样多于30%。除此之外，只要我们细心发掘，还可以挖掘出更多的潜在数据，而通过分析这些数据，则必须要获取到这些用户数据。此时，我们可以使用网络爬虫技术轻松爬取到这些有用的用户

同样，在2015年，有网友爬取了3000万QQ空间的用户信息。

通过前面章节的学习，我们已经基本认识了网络爬虫，那么网络爬虫应该怎么实现？核心技术又有哪些呢？在本篇中，我们首先会介绍网络爬虫的相关实现原理与实现技术；随后，讲解Urllib库的相关实战内容；紧接着，带领大家一起开发几种典型的网络爬虫，让大家在实战项目中由浅入深地掌握Python网络爬虫的开发；在学会了一些经典的网络爬虫开发之后，我们将一起研究学习Fiddler抓包分析技术、浏览器伪装技术、爬虫定向抓取技术等知识，让大家更加深入地进入到网络爬虫技术的世界中来。

除了以上两个例子之外，用户爬虫还可以做很多事情，比如爬取淘宝的用户信息，可以分析淘宝用户喜欢什么商品，从而更有利于我们对商品的定位等。由此看来，网络爬虫还可以获得很多有趣的潜在信息，那么这些爬虫难吗？其实不难，在阅读完本书后，相信你也能够写出这样的爬虫。

## 2.4 小结

1) 爬虫的出现，可以在一定程度上代替手工访问网页，所以，原先我们需要人工去访问互联网信息的行为，现在都可以用爬虫自动化实现，这样可以更高效地利用互联网中的有效信息。

2) 检索是一种行为，而索引是一种属性。检索是有一个好的结果，检索的效率取决于索引。若没有索引，则检索的效率会很低。

3) 用户爬虫是网络爬虫的一种，其原理与网络爬虫类似，只是爬取的是用户数据。由于互联网中用户数据量巨大，因此用户爬虫的爬取效率相对较低。

第1章 网络爬虫概述  
第2章 网络爬虫的组成  
第3章 网络爬虫的爬取原理  
第4章 网络爬虫的爬取技术  
第5章 网络爬虫的爬取工具  
第6章 网络爬虫的爬取应用  
第7章 网络爬虫的爬取进阶  
第8章 网络爬虫的爬取高级应用  
第9章 网络爬虫的爬取综合应用

## 网络爬虫实现原理与实现技术

我们已经初步认识了网络爬虫，并了解了网络爬虫的应用领域。在这一章中，我们将学习网络爬虫的实现原理及其实现技术，并使用 metaseeker 为大家做一个简单的爬虫案例。

### 3.1 网络爬虫实现原理详解

不同类型的网络爬虫，其实现原理也是不同的，但这些实现原理中，会存在很多共性。在此，我们将以两种典型的网络爬虫为例（即通用网络爬虫和聚焦网络爬虫），分别为大家讲解网络爬虫的实现原理。

#### 1. 通用网络爬虫

首先我们来看通用网络爬虫的实现原理。通用网络爬虫的实现原理及过程可以简要概括如下（见图 3-1）。

1) 获取初始的 URL。初始的 URL 地址可以由用户人为地指定，也可以由用户指定的某个或某几个初始爬取网页决定。

2) 根据初始的 URL 爬取页面并获得新的 URL。获得初始的 URL 地址之后，首先需要爬取对应 URL 地址中的网页，爬取了对应的 URL 地址中的网页后，将网页存储到原始数据库中，并且在爬取网页的同时，发现新的 URL 地址，同时将已爬取的 URL 地址存放到一个 URL 列表中，用于去重及判断爬取的进程。

3) 将新的 URL 放到 URL 队列中。在第 2 步中，获取了下一个新的 URL 地址之后，会将新的 URL 地址放到 URL 队列中。

4) 从 URL 队列中读取新的 URL，并依据新的 URL 爬取网页，同时从新网页中获取新

URL，并重复上述的爬取过程。

5) 满足爬虫系统设置的停止条件时，停止爬取。在编写爬虫的时候，一般会设置相应的停止条件。如果没有设置停止条件，爬虫则会一直爬取下去，一直到无法获取新的 URL 地址为止，若设置了停止条件，爬虫则会在停止条件满足时停止爬取。

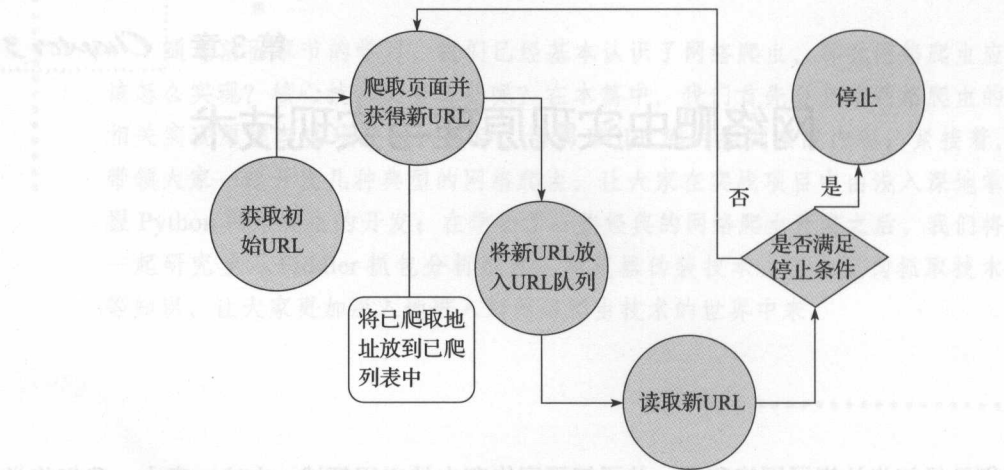


图 3-1 通用网络爬虫的实现原理及过程

以上就是通用网络爬虫的实现过程与基本原理，接下来，我们为大家分析聚焦网络爬虫的基本原理及其实现过程。

2. 聚焦网络爬虫

聚焦网络爬虫，由于其需要有目的地进行爬取，所以对于通用网络爬虫来说，必须要增加目标的定义和过滤机制，具体来说，此时，其执行原理和过程需要比通用网络爬虫多出三步，即目标的定义、无关链接的过滤、下一步要爬取的 URL 地址的选取等，如图 3-2 所示。

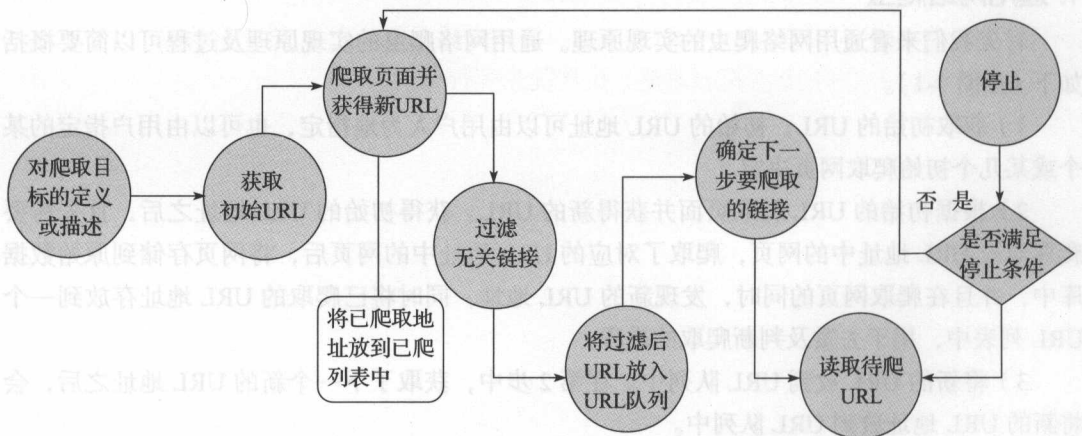


图 3-2 聚焦网络爬虫的基本原理及其实现过程

1) 对爬取目标的定义和描述。在聚焦网络爬虫中,我们首先要依据爬取需求定义好该聚焦网络爬虫爬取的目标,以及进行相关的描述。

2) 获取初始的 URL。

3) 根据初始的 URL 爬取页面,并获得新的 URL。

4) 从新的 URL 中过滤掉与爬取目标无关的链接。因为聚焦网络爬虫对网页的爬取是有目的性的,所以与目标无关的网页将会被过滤掉。同时,也需要将已爬取的 URL 地址存放到一个 URL 列表中,用于去重和判断爬取的进程。

5) 将过滤后的链接放到 URL 队列中。

6) 从 URL 队列中,根据搜索算法,确定 URL 的优先级,并确定下一步要爬取的 URL 地址。在通用网络爬虫中,下一步爬取哪些 URL 地址,是不太重要的,但是在聚焦网络爬虫中,由于其具有目的性,故而下一步爬取哪些 URL 地址相对来说是比较重要的。对于聚焦网络爬虫来说,不同的爬取顺序,可能导致爬虫的执行效率不同,所以,我们需要依据搜索策略来确定下一步需要爬取哪些 URL 地址。

7) 从下一步要爬取的 URL 地址中,读取新的 URL,然后依据新的 URL 地址爬取网页,并重复上述爬取过程。

8) 满足系统中设置的停止条件时,或无法获取新的 URL 地址时,停止爬行。

现在我们初步掌握了网络爬虫的实现原理以及相应的工作流程,下面来了解网络爬虫的爬行策略。

## 3.2 爬行策略

在网络爬虫爬取的过程,在待爬取的 URL 列表中,可能有很多 URL 地址,那么这些 URL 地址,爬虫应该先爬取哪个,后爬取哪个呢?在通用网络爬虫中,虽然爬取的顺序并不是那么重要,但是在其他很多爬虫中,比如聚焦网络爬虫中,爬取的顺序非常重要,而爬取的顺序,一般由爬行策略决定。在这一节中,我们将为大家介绍一些常见的爬行策略。

爬行策略主要有深度优先爬行策略、广度优先爬行策略、大站优先策略、反链策略、其他爬行策略等。下面我们将分别进行介绍。

如图 3-3 所示,假设有一个网站,ABCDEFGF 分别为站点下的网页,图中箭头表示网页的层次结构。

假如此时网页 ABCDEFG 都在爬行队列中,那么按照不同的爬行策略,其爬取的顺序是不同的。

比如,如果按照深度优先爬行策略去爬取的话,那么此时会首先爬取一个网页,然后将这个网页的下层链接依次深入爬取完再返回上一层进行爬取。

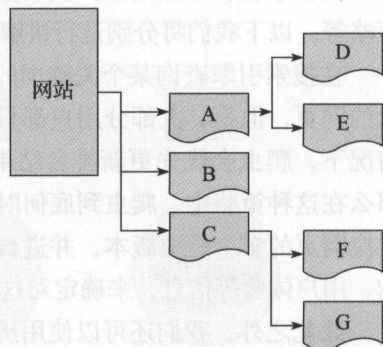


图 3-3 某网站的网页层次结构示意图



所以，若按深度优先爬行策略，图 3-3 中的爬行顺序可以是： $A \rightarrow D \rightarrow E \rightarrow B \rightarrow C \rightarrow F \rightarrow G$ 。

如果按照广度优先的爬行策略去爬取的话，那么此时首先会爬取同一层次的网页，将同一层次的网页全部爬取完后，在选择下一个层次的网页去爬行，比如，上述的网站中，如果按照广度优先的爬行策略去爬取的话，爬行顺序可以是： $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G$ 。

除了以上两种爬行策略之外，我们还可以采用大站爬行策略。我们可以按对应网页所属的站点进行归类，如果某个网站的网页数量多，那么我们则将其称为大站，按照这种策略，网页数量越多的网站越大，然后，优先爬取大站中的网页 URL 地址。

一个网页的反向链接数，指的是该网页被其他网页指向的次数，这个次数在一定程度上代表着该网页被其他网页的推荐次数。所以，如果按反链策略去爬行的话，那么哪个网页的反链数量越多，则哪个网页将被优先爬取。但是，在实际情况中，如果单纯按反链策略去决定一个网页的优先程度的话，那么可能会出现大量的作弊情况。比如，做一些垃圾站群，并将这些网站互相链接，如果这样的话，每个站点都将获得较高的反链，从而达到作弊的目的。作为爬虫项目方，我们当然不希望受到这种作弊行为的干扰，所以，如果采用反向链接策略去爬取的话，一般会考虑可靠的反链数。

除了以上这些爬行策略，在实际中还有很多其他的爬行策略，比如 OPIC 策略、Partial PageRank 策略等。

### 3.3 网页更新策略

一个网站的网页经常会更新，作为爬虫方，在网页更新后，我们则需要对这些网页进行重新爬取，那么什么时候去爬取合适呢？如果网站更新过慢，而爬虫爬取得过于频繁，则必然会增加爬虫及网站服务器的压力，若网站更新较快，但是爬虫爬取的时间间隔较长，则我们爬取的内容版本会过老，不利于新内容的爬取。显然，网站的更新频率与爬虫访问网站的频率越接近，则效果越好，当然，爬虫服务器资源有限的时候，此时爬虫也需要根据对应策略，让不同的网页具有不同的更新优先级，优先级高的网页更新，将获得较快的爬取响应。

具体来说，常见的网页更新策略主要有 3 种：用户体验策略、历史数据策略、聚类分析策略等，以下我们将分别进行讲解。

在搜索引擎查询某个关键词的时候，会出现一个排名结果，在排名结果中，通常会有大量的网页，但是，大部分用户都只会关注排名靠前的网页，所以，在爬虫服务器资源有限的情况下，爬虫会优先更新排名结果靠前的网页。这种更新策略，我们称之为用户体验策略，那么在这种策略中，爬虫到底何时去爬取这些排名结果靠前的网页呢？此时，爬取中会保留对应网页的多个历史版本，并进行对应分析，依据这多个历史版本的内容更新、搜索质量影响、用户体验等信息，来确定对这些网页的爬取周期。

除此之外，我们还可以使用历史数据策略来确定对网页更新爬取的周期。比如，我们可以依据某一个网页的历史更新数据，通过泊松过程进行建模等手段，预测该网页下一次更新



的时间,从而确定下一次对该网页爬取的时间,即确定更新周期。

以上两种策略,都需要历史数据作为依据。有的时候,若一个网页为新网页,则不会有对应的历史数据,并且,如果要依据历史数据进行分析,则需要爬虫服务器保存对应网页的历史版本信息,这无疑给爬虫服务器带来了更多的压力和负担。如果想要解决这些问题,则需要采取新的更新策略。比较常用的是聚类分析策略。那么什么是聚类分析策略呢?

在生活中,相信大家对分类已经非常熟悉,比如我们去商场,商场中的商品一般都分好类了,方便顾客去选购相应的商品,此时,商品分类的类别是固定的,是已经拟定好的。但是,假如商品的数量巨大,事先无法对其进行分类,或者说,根本不知道将会拥有哪些类别的商品,此时,我们应该如何解决将商品归类的问题呢?

这时候我们可以用聚类的方式解决,依据商品之间的共性进行相应分析,将共性较多的商品聚为一类,此时,商品聚集成的类的数目是不一定的,但是能保证的是,聚在一起的商品之间一定有某种共性,即依据“物以类聚”的思想去实现。

同样,在我们的聚类算法中,也会有类似的分析过程。

将聚类分析算法运用在爬虫对网页的更新上,我们可以这样做,如图3-4所示。

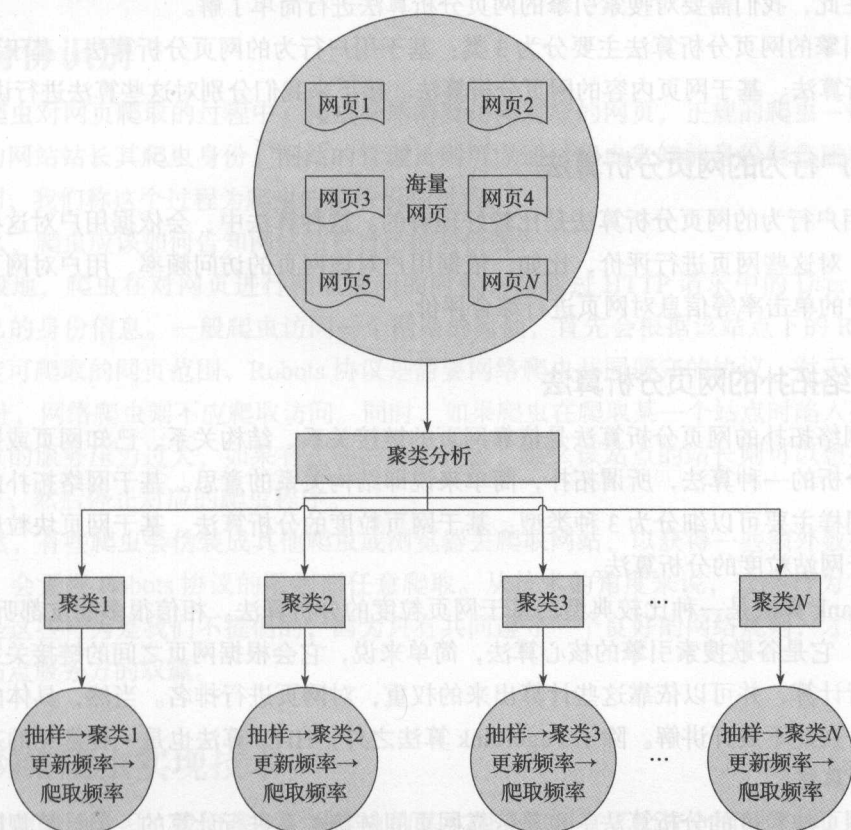


图3-4 网页更新策略之聚类算法

1) 首先, 经过大量的研究发现, 网页可能具有不同的内容, 但是一般来说, 具有类似属性的网页, 其更新频率类似。这是聚类分析算法运用在爬虫网页的更新上的一个前提指导思想。

2) 有了 1 中的指导思想后, 我们可以首先对海量的网页进行聚类分析, 在聚类之后, 会形成多个类, 每个类中的网页具有类似的属性, 即一般具有类似的更新频率。

3) 聚类完成后, 我们可以对同一个聚类中的网页进行抽样, 然后求该抽样结果的平均更新值, 从而确定对每个聚类的爬行频率。

以上, 就是使用爬虫爬取网页的时候, 常见的 3 种更新策略, 我们掌握了其算法思想后, 在后续我们进行爬虫的实际开发的时候, 编写出来的爬虫执行效率会更高, 并且执行逻辑会更合理。

### 3.4 网页分析算法

在搜索引擎中, 爬虫爬取了对应的网页之后, 会将网页存储到服务器的原始数据库中, 之后, 搜索引擎会对这些网页进行分析并确定各网页的重要性, 即会影响用户检索的排名结果。

所以在此, 我们需要对搜索引擎的网页分析算法进行简单了解。

搜索引擎的网页分析算法主要分为 3 类: 基于用户行为的网页分析算法、基于网络拓扑的网页分析算法、基于网页内容的网页分析算法。接下来我们分别对这些算法进行讲解。

#### 1. 基于用户行为的网页分析算法

基于用户行为的网页分析算法是比较好理解的。这种算法中, 会依据用户对这些网页的访问行为, 对这些网页进行评价, 比如, 依据用户对该网页的访问频率、用户对网页的访问时长、用户的单击率等信息对网页进行综合评价。

#### 2. 基于网络拓扑的网页分析算法

基于网络拓扑的网页分析算法是依靠网页的链接关系、结构关系、已知网页或数据等对网页进行分析的一种算法, 所谓拓扑, 简单来说即结构关系的意思。基于网络拓扑的网页分析算法, 同样主要可以细分为 3 种类型: 基于网页粒度的分析算法、基于网页块粒度的分析算法、基于网站粒度的分析算法。

PageRank 算法是一种比较典型的基于网页粒度的分析算法。相信很多朋友都听过 PageRank 算法, 它是谷歌搜索引擎的核心算法, 简单来说, 它会根据网页之间的链接关系对网页的权重进行计算, 并可以依靠这些计算出来的权重, 对网页进行排名。当然, 具体的算法细节有很多, 在此不展开讲解。除了 PageRank 算法之外, HITS 算法也是一种常见的基于网页粒度的分析算法。

基于网页块粒度的分析算法, 也是依靠网页间链接关系进行计算的, 但计算规则有所不同。我们知道, 在一个网页中通常会包含多个超链接, 但一般其指向的外部链接中并不是所

有的链接都与网站主题相关,或者说,这些外部链接对该网页的重要程度是不一样的,所以若要基于网页块粒度进行分析,则需要对一个网页中的这些外部链接划分层次,不同层次的外部链接对于该网页来说,其重要程度不同。这种算法的分析效率和准确率,会比传统的算法好一些。

基于网站粒度的分析算法,也与 PageRank 算法类似,但是,如果采用基于网站粒度进行分析,相应的,会使用 SiteRank 算法。即此时我们会划分站点的层次和等级,而不再具体地计算站点下的各个网页的等级。所以其相对于基于网页粒度的算法来说,则更加简单高效,但是会带来一些缺点,比如精确度不如基于网页粒度的分析算法精确。

### 3. 基于网页内容的网页分析算法

在基于网页内容的网页分析算法中,会依据网页的数据、文本等网页内容特征,对网页进行相应的评价。

以上,我简单为大家介绍了搜索引擎中的网页分析算法,我们学习爬虫,需要对这些算法进行相应的了解。

## 3.5 身份识别

在爬虫对网页爬取的过程中,爬虫必然需要访问对应的网页,正规的爬虫一般会告诉对应网页的网站站长其爬虫身份。网站的管理员则可以通过爬虫告知的身份信息对爬虫的身份进行识别,我们称这个过程为爬虫的身份识别过程。

那么,爬虫应该如何告知网站站长自己的身份呢?

一般地,爬虫在对网页进行爬取访问的时候,会通过 HTTP 请求中的 User Agent 字段告知自己的身份信息。一般爬虫访问一个网站的时候,首先会根据该站点下的 Robots.txt 文件来确定可爬取的网页范围,Robots 协议是需要网络爬虫共同遵守的协议,对于一些禁止的 URL 地址,网络爬虫则不应爬取访问。同时,如果爬虫在爬取某一个站点时陷入死循环,造成该站点的服务压力过大,如果有正确的身份设置,那么该站点的站长则可以想办法联系到该爬虫方,然后停止对应的爬虫程序。

当然,有些爬虫会伪装成其他爬虫或浏览器去爬取网站,以获得一些额外数据,或者有些爬虫,会无视 Robots 协议的限制而任意爬取。从技术的角度来说,这些行为实现起来不难,但是这些行为是我们不提倡的,因为只有共同遵守一个良好的网络规则,才能够达到爬虫方和站点服务方的双赢。

## 3.6 网络爬虫实现技术

通过前面的学习,我们基本上对爬虫的基本理论知识有了比较全面的了解,那么,如果我们要实现网络爬虫技术,要开发自己的网络爬虫,可以使用哪些语言进行开发呢?

开发网络爬虫的语言有很多，常见的语言有：Python、Java、PHP、Node.JS、C++、Go 语言等。以下我们将分别介绍一下用这些语言写爬虫的特点：

❑ Python：爬虫框架非常丰富，并且多线程的处理能力较强，并且简单易学、代码简洁，优点很多。

❑ Java：适合开发大型爬虫项目。

❑ PHP：后端处理很强，代码很简洁，模块也较丰富，但是并发能力相对来说较弱。

❑ Node.JS：支持高并发与多线程处理。

❑ C++：运行速度快，适合开发大型爬虫项目，成本较高。

❑ Go 语言：同样高并发能力非常强。

以上分别介绍了写爬虫的常见实现技术，本书中，笔者将会以 Python 语言为例，带领大家一步步地学好爬虫的开发。

### 3.7 实例——metaseeker

metaseeker 是一款比较实用的网站数据采集程序，使用该采集程序，可以让大家比较快速、形象地了解爬虫的工作过程。所以在本节中，会以 metaseeker 为例，跟大家一起学习如何采集当当网的商品及价格信息，让大家对爬虫工作过程有一个形象地了解，为后续我们使用 Python 开发爬虫打下基础。

如图 3-5 所示，我们将为大家爬取当当网新书栏目下的商品的名称及价格等信息（[http://e.dangdang.com/morelist\\_page.html?columnType=all\\_rec\\_xssf&title=%E6%96%B0%E4%B9%A6%E9%A6%96%E5%8F%91](http://e.dangdang.com/morelist_page.html?columnType=all_rec_xssf&title=%E6%96%B0%E4%B9%A6%E9%A6%96%E5%8F%91)）。



图 3-5 当当网新书网页



可以从官网下载 metaseeker 工具 (<http://www.gooseeker.com/pro/product.html>), 进入后, 选择第三种方案下载, 如图 3-6 所示。该软件有的版本可以与浏览器配合使用, 方案三集成了浏览器和该爬虫软件, 安装起来比较简单。

下载之后, 我们只需要打开安装即可, 安装好之后, 打开该软件, 会出现一个类似浏览器的界面, 我们打开要爬取的网址 (即刚才提到的当当网的图书商品页), 单击“MS 谋数台”, 如图 3-7 所示。

打开后, 会出现图 3-8 所示的界面。

此时, 我们需要将刚才的商品页面网址复制到左上角的网址处, 并按一下回车键, 如图 3-9 所示。在加载了一会儿之后, 软件的左下角处会出现“完成”字样, 此时代表网页加载完成。

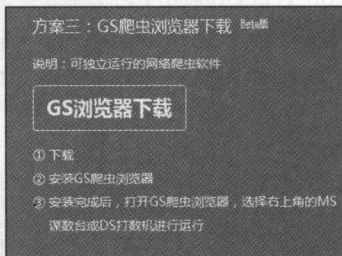


图 3-6 使用第三种方案下载

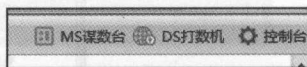


图 3-7 单击“MS 谋数台”

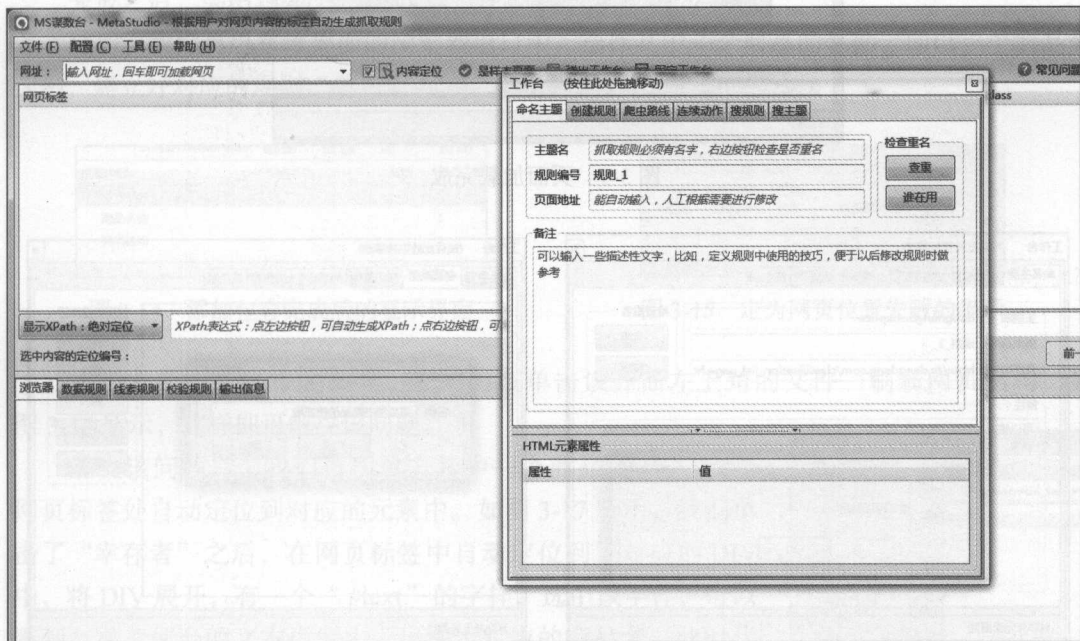


图 3-8 MS 谋数台显示界面

然后, 我们需要在该界面的“工作台”中, 创建命名主题, 创建好命名主题后, 需要单击“查重”按钮, 看是否名称冲突, 若名称冲突则需要换一个主题名字。如图 3-10 所示, 创建了一个名为 dangdangbookprice 的主题名。

创建主题名之后, 需要进行下一步操作, 即创建规则。我们在创建规则的页面中, 单击新建, 便可以输入想创建的规则名称, 该规则名称可以自己拟定, 如图 3-11 所示, 我们创建





随后,会出现如图 3-13 所示的界面,让我们填写被爬取内容的详细信息,此时,我们需要根据自己的需求规划好一共需要多少个包容,比如,在此我们需要爬取商品的名称和商品的价格,所以两个包容就够了。我们先创建第一个包容,即商品价格,输入对应名称,然后勾选好右边的“关键内容”。完成之后,可以单击保存,然后再次选中规则名,并右键添加第二个包容,即商品名称。

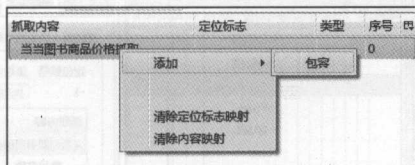


图 3-12 添加包容信息

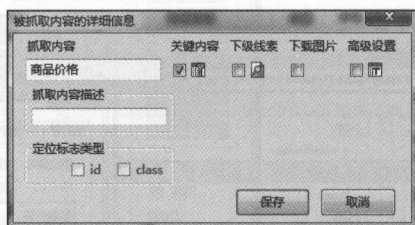


图 3-13 填写包容的详细信息

完成之后,会出现如图 3-14 所示界面。

随后,我们可以在该界面的浏览器窗口中,选择其中一个商品的名称,即以一个商品名为例,建立好对应的规则。单击后可能会出现如图 3-15 所示的提示。

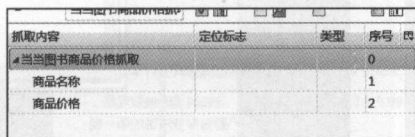


图 3-14 添加包容完成后的显示界面

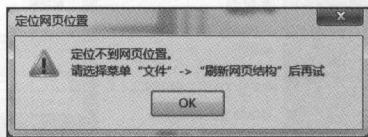


图 3-15 定为网页位置失败的提示

如果出现图 3-15 中的提示,我们可以单击该界面左上角的文件→刷新网页结构,如图 3-16 所示,这样即可解决该问题。

解决该问题后,我们再次单击其中一个商品名,单击后,网页标签处自动定位到对应的元素中。如图 3-17 所示,我们单击了“幸存者”之后,在网页标签中自动定位到了对应的 DIV 中,将 DIV 展开,有一个“#text”的字样,选中该字样,可以看到,在工作台的文本内容中,出现了对应的商品名,此时代表商品名定位成功。

随后,我们选中对应的“#text”,然后右键,单击内容映射→商品名称,将该规则映射到对应的商品名称包容中,那么以后,便可以根据这个规则去爬取网页上的其他商品的名称了,如图 3-18 所示。

我们还需要指定价格的规则,此时我们在浏览器区域中,单击该商品对应的价格,然后,在网页标签处会进行自动定位,如图 3-19 所示,我们单击了对应的价格“9.09”之

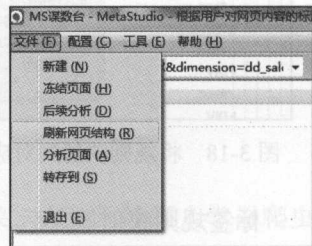


图 3-16 刷新网页结构

后，网页标签处，自动定位到了对应的 I 标签下，我们展开 I 标签，同样可以看得到一个“#text”，选中“#text”在工作台的文本内容中，会出现对应的价格信息，此时，代表定位成功。

然后，我们同样需要选中该标签，然后右击，将该标签映射到商品价格中，如图 3-20 所示。

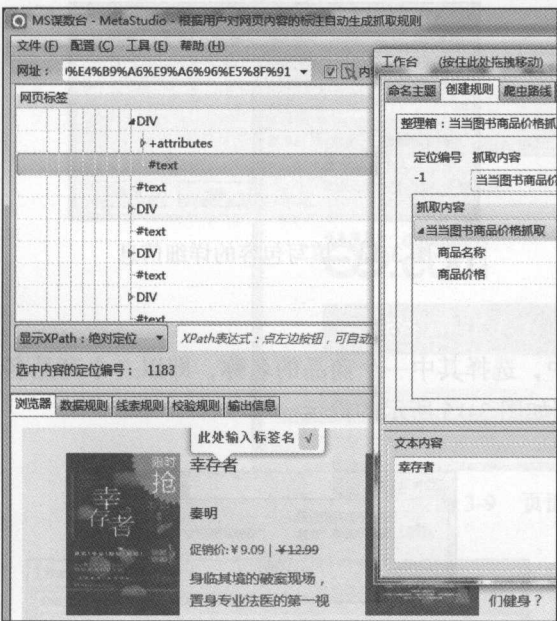


图 3-17 商品定位成功

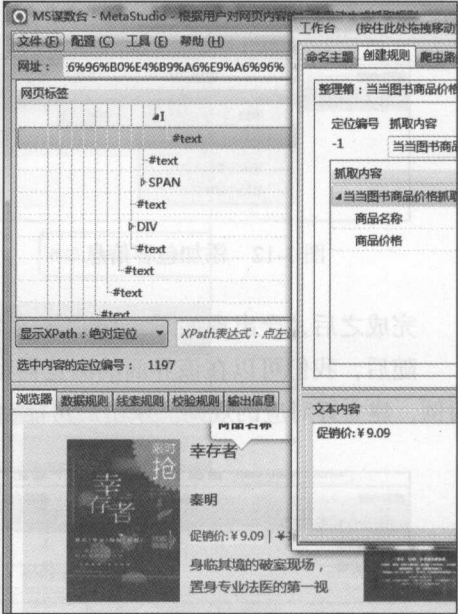


图 3-19 单击商品价格自动定位

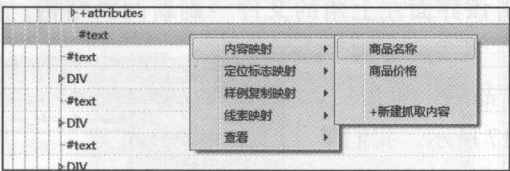


图 3-18 将规则映射到对应的商品名称包内容中

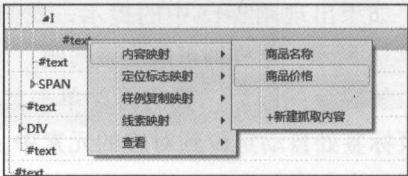


图 3-20 将标签映射到商品价格中

标签规则映射好之后，我们返回工作台，然后单击测试，便可以看得到当前是否爬取该界面中的所有商品信息。如图 3-21 所示，返回工作台，并单击测试。

单击了测试后，在输出信息中，我们可以看得到，该输出信息包含了该界面中所有的商品名称和对应的商品价格，也就是说，我们成功采集了，如图 3-22 所示，由于界面空间有限，只展现了部分爬取信息。

如果我们要对该网站下其他网页中的商品信息都进行自动爬取，虽然也是可以的，但是需要设置对应的爬取规则。在这里，metaseeker 的使用仅作为本书的一个实例，并不是本

书的重点内容,所以,关于 metaseeker 的深入使用部分我们就不过多讲解了,对应的内容不难,有兴趣的读者可以查看相关资料。

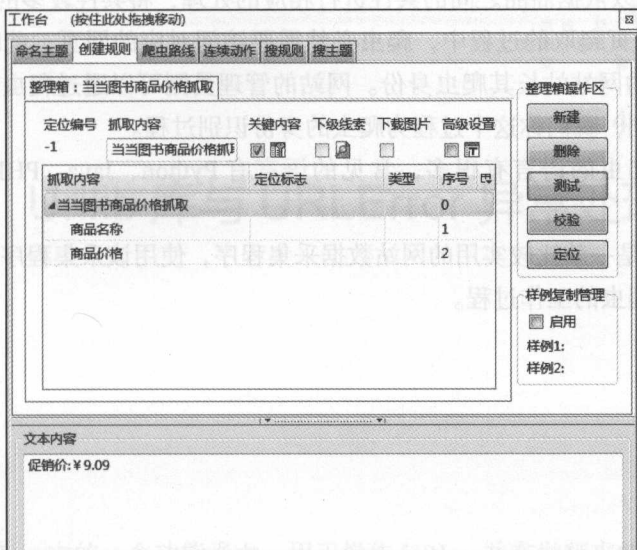


图 3-21 返回工作台

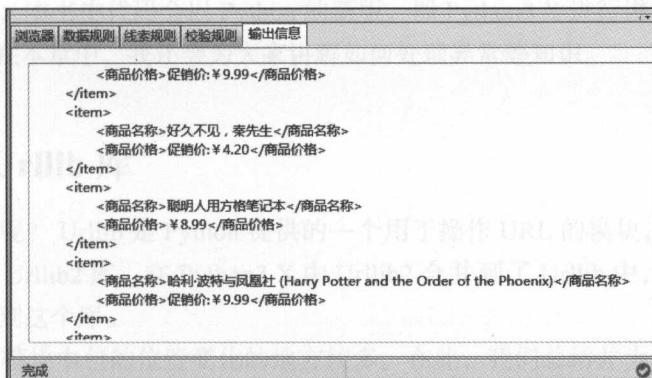


图 3-22 成功采集界面中所有的商品信息

我们讲该实例的目的是让大家对爬虫有一个形象的初步印象,方便后续深入学习爬虫开发。

### 3.8 小结

1) 聚焦网络爬虫,由于其需要有目的地进行爬取,所以对于通用网络爬虫来说,必须要增加目标的定义和过滤机制,具体来说,此时,其执行原理和过程需要比通用网络爬虫多



出 3 步, 即目标的定义、无链接的过滤、下一步要爬取的 URL 地址的选取。

2) 常见的网页更新策略主要有 3 种: 用户体验策略、历史数据策略、聚类分析策略。

3) 聚类分析可以依据商品之间的共性进行相应的处理, 将共性较多的商品聚为一类。

4) 在爬虫对网页爬取的过程中, 爬虫必然需要访问对应的网页, 此时, 正规的爬虫一般会告诉对应网页的网站站长其爬虫身份。网站的管理员则可以通过爬虫告知的身份信息对爬虫的身份进行识别, 我们称这个过程为爬虫的身份识别过程。

5) 开发网络爬虫的语言有很多, 常见的语言有 Python、Java、PHP、Node.JS、C++、Go 语言等。

6) metaseeker 是一款比较实用的网站数据采集程序, 使用该采集程序, 可以让大家比较快速、形象地了解爬虫的工作过程。

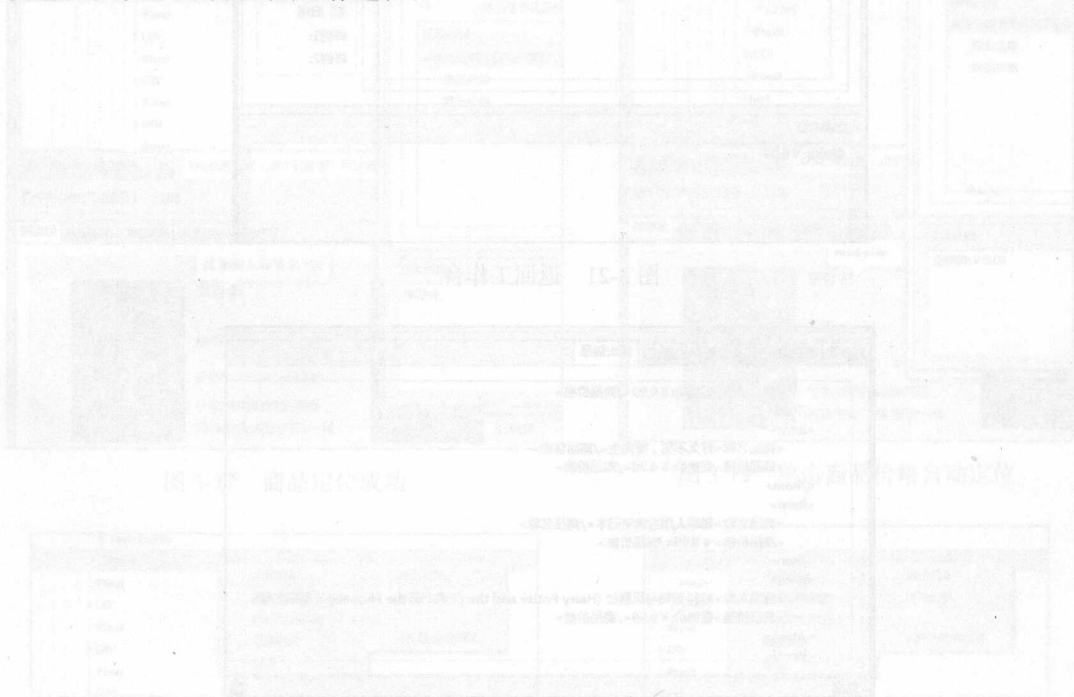


图 3-18 商品定位成功

图 3-18 将爬取到的商品数据写入到数据库中

爬虫程序运行成功后, 爬虫程序会自动将爬取到的数据写入到数据库中, 如图 3-21 所示, 返回了爬取结果, 并显示了爬取结果。

单击了爬取按钮, 在输出信息中, 我们可以得到, 该地址信息包含了该网站中所有的商品名称和对应的商品价格, 也就是说, 我们成功采集了, 如图 3-22 所示, 显示了爬取结果。

爬虫程序运行成功后, 爬虫程序会自动将爬取到的数据写入到数据库中, 如图 3-21 所示, 返回了爬取结果, 并显示了爬取结果。



## Urllib 库与 URLError 异常处理

Urllib 库是 Python 中的一个功能强大、用于操作 URL，并在做爬虫的时候经常要用到的库。在 Python2.X 中，分 Urllib 库和 Urllib2 库，Python3.X 之后合并到 Urllib 库中，使用方法稍有不同，在此，本书中代码会以 Python 的新版，即 Python3.X 进行讲解，具体使用的是 Python3.5.2。同时在本章中，我还会为大家讲解如何处理异常等知识。

### 4.1 什么是 Urllib 库

什么是 Urllib 呢？Urllib 是 Python 提供的一个用于操作 URL 的模块，在 Python2.X 中，有 Urllib 库，也有 Urllib2 库，在 Python3.X 中 Urllib2 合并到了 Urllib 中，我们爬取网页的时候，经常需要用到这个库。

升级合并后，模块中包的位置变化的地方较多。在此，我们总结并为大家列举一些常见的位置变动，方便之前用 Python2.X 的朋友在使用 Python3.X 的时候可以快速掌握，其他的我们在用到时具体为大家讲解。

常见的变化有：

- ❑ 在 Python2.X 中使用 `import urllib2`——对应的，在 Python3.X 中会使用 `import urllib.request, urllib.error`。
- ❑ 在 Python2.X 中使用 `import urllib`——对应的，在 Python3.X 中会使用 `import urllib.request, urllib.error, urllib.parse`。
- ❑ 在 Python2.X 中使用 `import urlparse`——对应的，在 Python3.X 中会使用 `import urllib.parse`。

- ❑ 在 Python2.X 中使用 `import urllib2`——对应的，在 Python3.X 中会使用 `import urllib.request, urllib.error`。
- ❑ 在 Python2.X 中使用 `urllib2.urlopen`——对应的，在 Python3.X 中会使用 `urllib.request.urlopen`。
- ❑ 在 Python2.X 中使用 `urllib.urlencode`——对应的，在 Python3.X 中会使用 `urllib.parse.urlencode`。
- ❑ 在 Python2.X 中使用 `urllib.quote`——对应的，在 Python3.X 中会使用 `urllib.request.quote`。
- ❑ 在 Python2.X 中使用 `cookielib.CookieJar`——对应的，在 Python3.X 中会使用 `http.CookieJar`。
- ❑ 在 Python2.X 中使用 `urllib2.Request`——对应的，在 Python3.X 中会使用 `urllib.request.Request`。

以上，总结了 Urllib 相关模块中从 Python2.X 到 Python3.X 的常见的一些变动，如果之前用的是 Python2.X 版本或者在网上阅读 Python2.X 关于 Urllib 这一块的代码，依据这个变动关系，可以快速写出 Python3.X 的程序。

如果没有 Urllib 这一方面的基础，那么没关系，在这一节中只需要了解 Urllib 库是做什么的即可，该变动关系可以暂时不用背下，在学习了本章后面几节之后，读者将详细了解这些代码的具体应用，懂得实战应用之后再记忆亦可，在以后遇到用 Python2.X 写的关于 Urllib 这一块的代码时，该变动关系同样会让你更快地理解一些常见的代码，解决 Python2.X 与 Python3.X 版本之间的阻碍。

## 4.2 快速使用 Urllib 爬取网页

以上我们对 Urllib 库做了简单的介绍，接下来，我将为大家讲解如何使用 Urllib 快速爬取一个网页。

要使用 Urllib 爬取网页，首先需要导入用到的对应模块，所以，我们可以输入如下代码导入 `urllib.request`：

```
>>> import urllib.request
```

在导入了模块之后，我们需要使用 `urllib.request.urlopen` 打开并爬取一个网页，此时，可以输入如下代码爬取百度首页（<http://www.baidu.com>），爬取后，将爬取到的网页赋给了变量 `file`：

```
>>> file=urllib.request.urlopen("http://www.baidu.com")
```

此时，我们还需要将对应的网页内容读取出来，可以使用 `file.read()` 读取全部内容，或者也可以使用 `file.file.readline()` 读取一行内容。执行如下代码：

```
>>> data=file.read()
>>> dataline=file.readline()
```

可以看到，我们分别读取了爬取到的网页的全部内容和一行内容，并分别赋给了变量 data, dataline。

随后，我们可以使用 print() 输出爬取到的内容，比如，如果要输出爬取的内容中的一行，可以输入：

```
>>> print(dataline)
```

执行后，可以看到如下所示的结果：

```
>>>print(dataline)
b"
```

此时，只输出了爬取到的内容中的第一行。我们可以输出全部内容：

```
>>> print(data)
```

执行后，可以看到如图 4-1 所示界面，这是成功将网页爬取下来后取得的网页的 HTML 代码。

```
>>>print(data)
b'<!DOCTYPE html><!--STATUS OK--><html><head><meta http-equiv="content-type" content="text/html; charset=utf-8"><meta http-equiv="X-UA-Compatible" content="IE=Edge"><meta content="always" name="referrer"><meta name="theme-color" content="#2932a1"><link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" /><link rel="search" type="application/opensearchdescription+xml" href="/content-search.xml" title="x7x99xbex5yxbax6x6x90x9cx7xb4xa2" /><link rel="icon" sizes="any" mask href="//www.baidu.com/img/baidu.svg"><link rel="dns-prefetch" href="//s1.bdstatic.com"/><link rel="dns-prefetch" href="//t1.baidu.com"/><link rel="dns-prefetch" href="//t2.baidu.com"/><link rel="dns-prefetch" href="//t3.baidu.com"/><link rel="dns-prefetch" href="//t10.baidu.com"/><link rel="dns-prefetch" href="//t11.baidu.com"/><link rel="dns-prefetch" href="//t12.baidu.com"/><link rel="dns-prefetch" href="//b1.bdstatic.com"/><title>x7x99xbex5yxbax6x4xb8x80x4xb8x8bxfefxbcx8cx4xbdx8a0x5xb0xb1x7x9fxa5x9x81x93</title><n<style index="index" id="css_index">html,body{height:100%;html{overflow-y:auto;}body{font:12px arial;text-align:center;background:#fff;}body,p,form,ul,li{margin:0;padding:0;}list-style:none;}body,form,#fm{position:relative;}td{text-align:left;}img{border:0;}a{color:#00c;a:active{color:#f60;}input{border:0;padding:0;}#wrapper{position:relative;position:min-height:100%;}#head{padding-bottom:100px;text-align:center;*z-index:1;}#ftCon{height:100px;position:absolute;bottom:23px;text-align:center;width:100%;margin:0 auto;z-index:0;overflow:hidden}.ftCon-Wrapper{overflow:hidden;ma
```

图 4-1 爬虫爬取的网页代码



**注意** 读取内容常见的有 3 种方式，其用法是：

- 1) file.read() 读取文件的全部内容，与 readlines 不同的是，read 会把读取到的内容赋给一个字符串变量。
- 2) file.readlines() 读取文件的全部内容，与 read 不同的是，readlines 会把读取到的内容赋给一个列表变量，若要读取全部内容，推荐使用这种方式。
- 3) File.readline() 读取文件的一行内容。

此时，我们已经成功实现了一个网页的爬取，那么我们如何将爬取到的网页以网页的形式保存在本地呢？

思路如下：

- 1) 首先，爬取一个网页并将爬取到的内容读取出来赋给一个变量。
- 2) 以写入的方式打开一个本地文件，命名为 \*.html 等网页格式。
- 3) 将 1) 中变量的值写入该文件中。
- 4) 关闭该文件。

所以，我们刚才已经成功获取到了百度首页的内容并读取赋给了变量 data，接下来，可以通过以下代码实现将爬取到的网页保存在本地。

```
>>> fhandle=open("D:/Python35/myweb/part4/1.html","wb")
>>> fhandle.write(data)
99437
>>> fhandle.close()
```

执行完该操作后，即将对应文件保存在 D:/Python35/myweb/part4 目录中了（当然在执行前先建立这个目录）。我们首先通过 open() 函数打开了该文件，并以“wb”即二进制写入的方式打开，打开后将句柄赋给变量 fhandle，然后使用 write() 方法写入了对应的数据 data，此时，我们看到写入的字节是 99437，写入后再通过 close() 方法关闭该文件，有始有终。

此时，在 D:/Python35/myweb/part4 目录下，可以看到该文件已经存在了，如图 4-2 所示，我们用浏览器打开该文件。

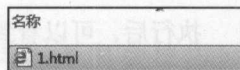


图 4-2 自动爬取并保存到本地的网页

用浏览器打开后，出现如图 4-3 所示的界面。

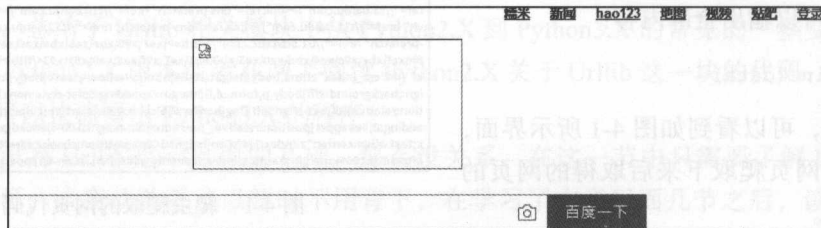


图 4-3 自动爬取并保存到本地的网页内容

此时，我们已经成功用程序自动地将百度首页的代码爬取到本地，只不过图片暂时没有爬取过来而已。

除了这种方法之外，我们还可以使用 urllib.request 里面的 urlretrieve() 函数直接将对应信息写入本地文件，格式为：“urllib.request.urlretrieve (url, filename = 本地文件地址)”。比如，我们可以直接使用该方式将网页写入本地文件，输入：

```
>>> filename=urllib.request.urlretrieve("http://edu.51cto.com", filename="D:/Python35/myweb/part4/2.html")
```

执行后，成功将网页“http://edu.51cto.com”（51CTO 学院）保存到了本地，我们去对应目录看，发现多出了 2.html 文件，如图 4-4 所示，该文件就是刚才代码自动保存的一个文件。

打开该网页，发现出现了 51CTO 学院首页的内容，如图 4-5 所示，说明成功爬取并保存到本地。

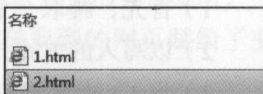


图 4-4 自动爬取并保存到本地的第二个网页

Urlretrieve 执行的过程中，会产生一些缓存，如果我们想清除这些缓存信息，可以使用 urlcleanup() 进行清除，输入如下代码即可清除 Urlretrieve 执行所造成的缓存：





图 4-5 自动爬取并保存到本地的第二个网页内容

```
>>> urllib.request.urlcleanup()
```

除此之外，urllib 中还有一些常见的用法，在此我们一并讲一下。

如果希望返回与当前环境有关的信息，我们可以使用 info() 返回，比如可以执行：

```
>>> file.info()
<http.client.HTTPMessage object at 0x0000000003623D68>
```

可以看到，输出了对应的 info，调用格式则为：“爬取的网页.info()”，我们之前爬取到的网页赋给了变量 file，所以此时通过 file 调用。

如果希望获取当前爬取网页的状态码，我们可以使用 getcode()，若返回 200 为正确，返回其他则不正确，调用格式则为：“爬取的网页.getcode()”。在该例中，我们可以执行：

```
>>> file.getcode()
200
```

可以看到，此时返回了状态码 200，说明此时响应正确。

如果想要获取当前所爬取的 URL 地址，我们可以使用 geturl() 来实现，调用格式则为：“爬取的网页.geturl()”，本例中，可以通过如下代码获取：

```
>>> file.geturl()
'http://www.baidu.com'
```

可以看到，此时输出了爬取的源网页地址为 'http://www.baidu.com'。

一般来说，URL 标准中只会允许一部分 ASCII 字符比如数字、字母、部分符号等，而其他的一些字符，比如汉字等，是不符合 URL 标准的。所以如果我们在 URL 中使用一些其他不符合标准的字符就会出现問題，此时需要进行 URL 编码方可解决。比如在 URL 中输入中文或者“:”或者“&”等不符合标准的字符时，需要编码。



如果要进行编码,我们可以使用 `urllib.request.quote()` 进行,比如,我们如果要对网址“`http://www.sina.com.cn`”进行编码,可以使用如下代码实现:

```
>>> urllib.request.quote("http://www.sina.com.cn")
'http%3A//www.sina.com.cn'
```

可以看到,编码的结果为 `'http%3A//www.sina.com.cn'`。

那么相应的,有时需要对编码的网址进行解码,应该怎么办呢?

若要解码,可以使用 `urllib.request.unquote()`,比如我们要对刚才编码的网址进行解码,可以输入:

```
>>> urllib.request.unquote("http%3A//www.sina.com.cn")
'http://www.sina.com.cn'
```

可以看到,解码后,成功恢复为原来的网址: `'http://www.sina.com.cn'`。

通过前面的学习,我们已经知道如何爬取一个网页的信息并进行简单的处理了,在了解这些基础知识之后接下来我们会对 `urllib` 库进行更深入的讲解。

## 4.3 浏览器的模拟——Headers 属性

有的时候,我们无法爬取一些网页,会出现 403 错误,因为这些网页为了防止别人恶意采集其信息所以进行了一些反爬虫的设置。

那么如果我们想爬取这些网页的信息,应该怎么办呢?

可以设置一些 Headers 信息,模拟成浏览器去访问这些网站,此时,就能够解决这个问题。

接下来我们一起来试试吧。

首先,可以使用上一节的方法爬取 CSDN 博客的内容:

```
>>> import urllib.request
>>> url= "http://blog.csdn.net/weiwei_pig/article/details/51178226"
>>> file=urllib.request.urlopen(url)
```

执行后,出现如图 4-6 所示错误:

可以看到,此时出现了 403 错误,即禁止访问的错误,所以接下来,我们需要让爬虫模拟成浏览器。我在之前理论部分为大家提到,模拟成浏览器可以设置 User-Agent 信息。

任意打开一个网页,比如打开百度首页 (`http://www.baidu.com`),然后按 F12,此时,会出现一个窗口,如图 4-7 所示,我们切换到 Network 标签页:

然后单击网页中的“百度一下”,如图 4-8 所示,即让网页发生一个动作。

此时,我们可以观察到下方的窗口出现了一些数据,如图 4-9 所示。

File "D:\Python35\lib\urllib\request.py", line 590, in http\_error\_default  
raise HTTPError(req.full\_url, code, msg, hdrs, fp)  
urllib.error.HTTPError: HTTP Error 403: Forbidden

图 4-6 出现 403 异常

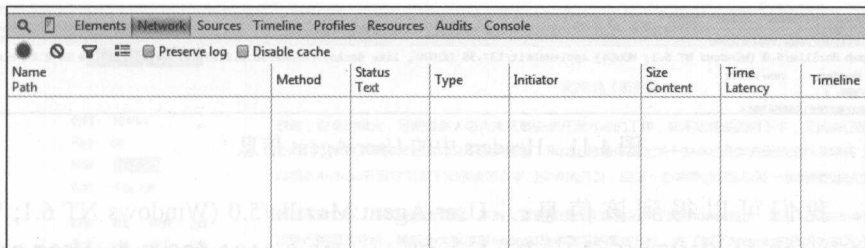


图 4-7 通过 F12 打开调试窗口



图 4-8 任意单击一个网页中的链接，使网页发生动作

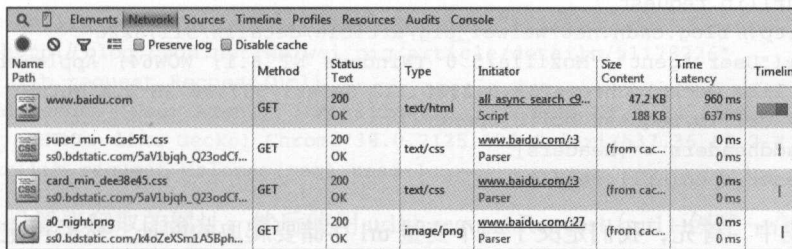


图 4-9 观察调试窗口中的数据变化

此时单击一下如图 4-9 中的“www.baidu.com”，即出现如图 4-10 所示的界面。

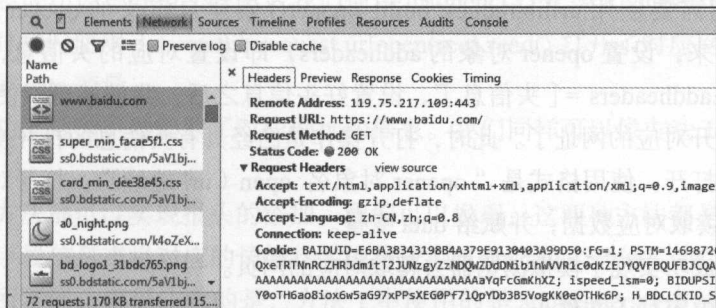


图 4-10 查看某个会话的 Headers 信息

将图 4-10 界面右方的标签切换到“Headers”中，即可以看到了对应的头信息，此时，往下拖动，可以找到 User-Agent 字样的一串信息，如图 4-11 所示，这一串信息即是我们下面模拟浏览器所需要用到的信息。我们将其复制出来。

Headers	Preview	Response	Cookies	Timing
Referer: https://www.baidu.com/ User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0				
▼ Response Headers	view source			
BDPAGETYPE: 2 BDQID: 8xc8d78cf178884384e				

图 4-11 Headers 中的 User-Agent 信息

此时，我们可以得到该信息：“User-Agent:Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0”。

接下来，我们讲解两种让爬虫模拟成浏览器访问网页的设置方法。

### 方法 1：使用 build\_opener() 修改报头

由于 urllib.open() 不支持一些 HTTP 的高级功能，所以，我们如果要修改报头，可以使用 urllib.request.build\_opener() 进行，比如，如果要爬取刚才无法爬取的网页，我们可以使用如下代码：

```
import urllib.request
url= "http://blog.csdn.net/weiwei_pig/article/details/51178226"
headers=("User-Agent","Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0")
opener = urllib.request.build_opener()
opener.addheaders = [headers]
data=opener.open(url).read()
```

上述代码中，首先，我们定义了一个变量 url 存储要爬取的网址，然后再定义一个变量 headers 存储对应的 User-Agent 信息，定义的格式为（“User-Agent”，具体信息），具体信息我们刚才已经从浏览器中获取了，该信息获取一次即可，以后在爬取其他网站的时候可以直接用，所以可以保存起来，不用每次都通过 F12 去找。

然后，我们需要使用 urllib.request.build\_opener() 创建自定义的 opener 对象并赋给变量 opener，接下来，设置 opener 对象的 addheaders，即设置对应的头信息，设置格式为：“opener 对象名.addheaders=[头信息]”，设置好头信息之后，我们就可以使用 opener 对象的 open() 方法打开对应的网址了。此时，打开操作是已经具有头信息的打开操作行为，即会模仿为浏览器去打开，使用格式是“opener 对象名.open(url 地址)”。打开对应网址后，再使用 read() 方法读取对应数据，并赋给 data 变量。

此时，我们成功实现了模拟浏览器去爬取对应的网页。

可以将对应内容写入文件：

```
>>> fhandle=open("D:/Python35/myweb/part4/3.html","wb")
>>> fhandle.write(data)
47630
>>> fhandle.close()
```

然后去对应文件夹打开 3.html，可以发现网页已经爬取成功，如图 4-12 所示。

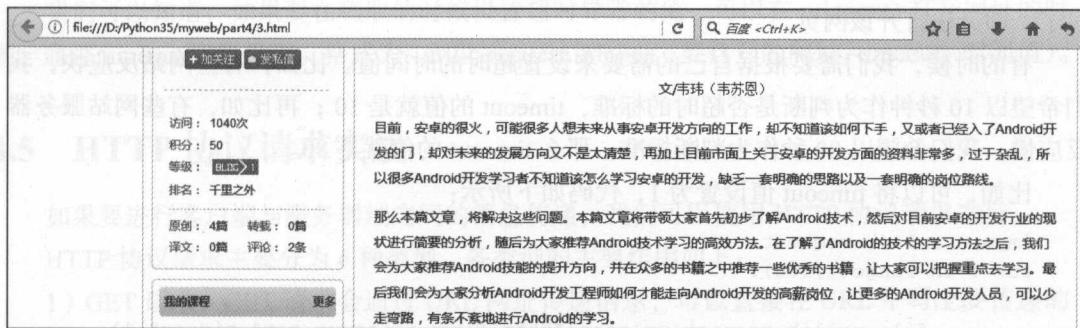


图 4-12 成功爬取结果图

### 方法 2: 使用 add\_header() 添加报头

除了上面这种方法之外, 还可以使用 `urllib.request.Request()` 下的 `add_header()` 实现浏览器的模拟。

比如, 我们可以通过如下代码实现:

```
import urllib.request
url= "http://blog.csdn.net/weiwei_pig/article/details/51178226"
req=urllib.request.Request(url)
req.add_header('User-Agent', 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0')
data=urllib.request.urlopen(req).read()
```

首先, 设置要爬取的网址, 然后使用 `urllib.request.Request (url)` 创建一个 `Request` 对象并赋给变量 `req`, 创建对象的格式为: `urllib.request.Request (url 地址)`。

随后, 使用 `add_header()` 方法添加对应的报头信息, 格式为: `Request 对象名.add_header (字段名, 字段值)`。

此时, 已经成功设置好报头, 然后我们使用 `urlopen()` 打开该 `Request` 对象即可打开对应网址, 所以此时我们使用 `data=urllib.request.urlopen(req).read()` 打开了对应网址并读取了网页内容, 并赋给了 `data` 变量。

此时, 成功模拟浏览器爬取了对应网址的信息。我们同样可以像方法 1 中一样将对应的信息写入文件。

以上两种方法都可以实现报头的添加, 我们可以发现, 这两种方法都是使用设置报头中的 `User-Agent` 字段信息来将对应的访问行为模仿成浏览器访问, 避免了 403 错误。只是添加报文的方法有所不同, 值得注意的是, 方法 1 中使用的是 `addheaders()` 方法, 方法 2 中使用的是 `add_header()` 方法, 注意末尾有无 `s` 以及有无下划线的区别。

## 4.4 超时设置

有的时候, 我们访问一个网页, 如果该网页长时间未响应, 那么系统就会判断该网页超



时了，即无法打开该网页。

有的时候，我们需要根据自己的需要来设置超时的时间值。比如，有些网站反应快，我们希望以 10 秒钟作为判断是否超时的标准，timeout 的值就是 10；再比如，有些网站服务器反应慢，我们希望以 80 秒作为判断标准，那么 timeout 的值就是 80。

比如，可以将 timeout 值设置为 1，代码如下所示：

```
import urllib.request
for i in range(1,100):
    try:
        file=urllib.request.urlopen("http://yum.iqianyue.com",timeout=1)
        data=file.read()
        print(len(data))
    except Exception as e:
        print(" 出现异常 -->" +str(e))
```

上述代码中，我们进行了 99 次循环，每次循环都会试着爬取网址“http://yum.iqianyue.com”，超时设置为 1 秒钟，即 1 秒钟未响应则判定为超时，并读取该网站的内容，输出获取到的内容的长度。

如果超时，则会引发异常，输出“出现异常”等字样，并输出对应的异常原因。

我们按 F5 执行之后，出现如图 4-13 所示的执行结果。

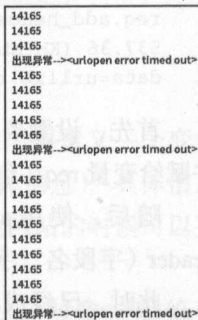
执行结果显示，在这 99 次循环中，有时能成功爬取网站内容并输出内容长度，有时却引发了异常，并且异常原因是超时。为什么会出现这种情况呢？

因为我们将 timeout 的值设置为 1 秒钟，相对来说，响应时间是比较短的，而我们在短时间内向该网站发送了大量访问请求，自然有的时候，服务器在 1 秒钟之内无法响应，但在大部分时候，网站可以在 1 秒钟之内作出响应，所以，我们看到，大部分时候是没有异常的，但频繁发送请求久了之后会引发异常。

尤其是在某些网站服务器性能不好的情况下，这种异常会更加频繁，如果不想出现这种异常，则可以将 timeout 的值按照需求写大一些，比如，以上的代码中，如果不想出现超时异常，我们可以将 timeout 的值设置为大一些（比如 30）即可，代码如下所示：

```
import urllib.request
for i in range(1,100):
    try:
        file=urllib.request.urlopen("http://yum.iqianyue.com",timeout=30)
        data=file.read()
        print(len(data))
    except Exception as e:
        print(" 出现异常 -->" +str(e))
```

按 F5 执行以上代码，没有发生超时异常。



```
14165
14165
14165
出现异常-->urlopen error timed out
14165
14165
14165
14165
14165
14165
出现异常-->urlopen error timed out
14165
14165
14165
14165
14165
14165
14165
14165
14165
14165
出现异常-->urlopen error timed out
```

图 4-13 有时会出现  
超时异常



我们可以知道,如果要在爬取的时候设置超时异常的值,可以在 `urlopen()` 打开网址的时候,通过 `timeout` 字段设置。格式为: `urllib.request.urlopen(要打开的网址, timeout=时间值)`。

## 4.5 HTTP 协议请求实战

如果要进行客户端与服务器端之间的消息传递,我们可以使用 HTTP 协议请求进行。

HTTP 协议请求主要分为 6 种类型,各类型的主要作用如下:

1) GET 请求: GET 请求会通过 URL 网址传递信息,可以直接在 URL 中写上要传递的信息,也可以由表单进行传递。如果使用表单进行传递,这表单中的信息会自动转为 URL 地址中的数据,通过 URL 地址传递。

2) POST 请求:可以向服务器提交数据,是一种比较主流也比较安全的数据传递方式,比如在登录时,经常使用 POST 请求发送数据。

3) PUT 请求:请求服务器存储一个资源,通常要指定存储的位置。

4) DELETE 请求:请求服务器删除一个资源。

5) HEAD 请求:请求获取对应的 HTTP 报头信息。

6) OPTIONS 请求:可以获得当前 URL 所支持的请求类型。

除此之外,还有 TRACE 请求与 CONNECT 请求等,TRACE 请求主要用于测试或诊断。由于用得非常少,所以这里不再提及。

接下来,我们将为大家通过实例讲解 HTTP 协议请求中的 GET 请求和 POST 请求,这两种请求相对来说用得最多。

### 1. GET 请求实例分析

有时想在百度上查询一个关键词,我们会打开百度首页,并输入该关键词进行查询,那么这个过程怎样使用爬虫自动实现呢?

我们首先需要对查询过程进行相应的分析,可以打开百度首页,然后输入想检索的关键词,比如输入“hello”,然后按回车键,此时会出现对应的查询结果,我们观察一下 URL 的变化,如图 4-14 所示。

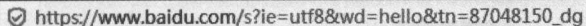


图 4-14 百度中搜索某个关键词时 URL 网址的变化

可以发现,对应的查询信息是通过 URL 传递的,这里说采用的是 HTTP 请求中的 GET 方法,我们将该网址提取出来就行分析,该网址为: `https://www.baidu.com/s?ie=utf8&wd=hello&tn=87048150_dg`,字段 `ie` 的值为 `utf8`,代表的是编码信息,而字段 `wd` 为 `hello`,刚好是我们要查询的信息,所以字段 `wd` 应该存储的就是用户带检索的关键词。根据我们的猜测,简化一下该网址,可以简化为: `https://www.baidu.com/s?wd=hello`,此时只

包含了对应的 wd 字段, 即待检索关键词字段, 将该网址复制到浏览器中, 刷新一下, 发现该网址也能够出现关键词为“hello”的搜索结果。由此可见, 我们在百度上查询一个关键词时, 会使用 GET 请求进行, 其中关键性字段是 wd, 网址的格式是: "https://www.baidu.com/s?wd= 关键词"。

根据分析出的这个规律, 即可以通过构造 GET 请求, 用爬虫实现在百度上自动查询某个关键词。

比如, 如果要实现用爬虫自动地在百度上查询关键词为 hello 的结果, 可以使用以下代码实现:

```
import urllib.request
keywd="hello"
url="http://www.baidu.com/s?wd="+keywd
req=urllib.request.Request(url)
data=urllib.request.urlopen(req).read()
fhandle=open("D:/Python35/myweb/part4/4.html","wb")
fhandle.write(data)
fhandle.close()
```

按 F5 执行了以上代码之后, 去 D:/Python35/myweb/part4/ 目录打开 4.html 网页, 出现如图 4-15 所示的结果, 说明已经成功构造 GET 请求, 并爬取了检索之后的网页, 这一切都是由程序自动完成的:

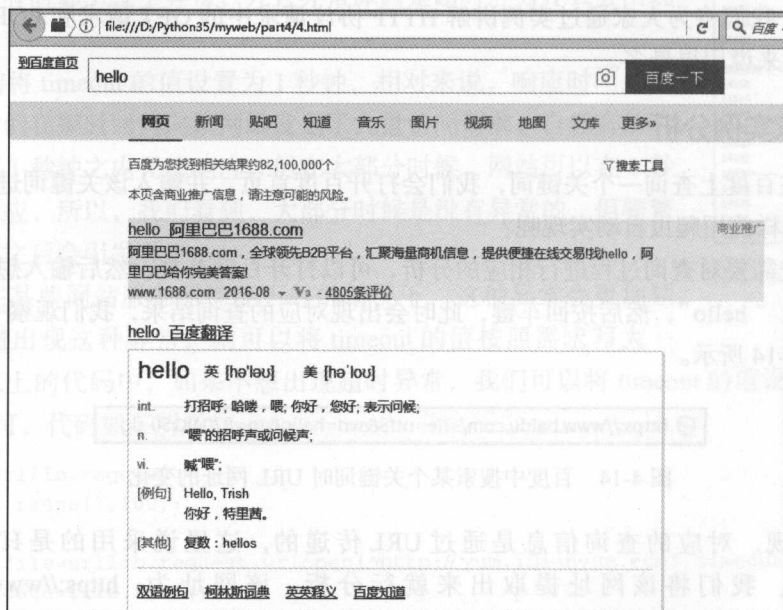


图 4-15 自动爬取并将结果保存到本地文件中, 此为爬取结果

**代码分析:** 接下来, 我们将详细分析一下上面的代码。首先定义要查询的关键词, 并赋

给 `keywd` 变量, 然后按照分析出来的 URL 格式, 构造了对应的 URL 并赋值给对应的 `url` 变量, 随后, 使用 `urllib.request.Request()` 构建了一个 `Request` 对象并赋给变量 `req`, 通过 `urllib.request.urlopen()` 打开对应的 `Request` 对象, 此时, 由于网址中包含了 GET 请求信息, 所以会以 GET 请求的方式获取该页面, 随后读取该页面的内容并赋值给 `data` 变量, 再将爬取到的内容写入 “D:/Python35/myweb/part4/4.html” 网页文件中。

上面的代码有不完善的地方, 如果我们要检索的关键词是中文, 如 “韦玮老师”, 若还以上面的代码去执行, 则会出现如下错误。

```
UnicodeEncodeError: 'ascii' codec can't encode characters in position 10-13:
ordinal not in range(128)
```

可以发现, 这是由编码问题造成的, 此时, 利用 4.2 节中的知识点即可解决该问题, 可以将上述代码进行以下优化:

```
import urllib.request
url="http://www.baidu.com/s?wd="
key=" 韦玮老师 "
key_code=urllib.request.quote(key)
url_all=url+key_code
req=urllib.request.Request(url_all)
data=urllib.request.urlopen(req).read()
fh=open("D:/Python35/myweb/part4/5.html", "wb")
fh.write(data)
fh.close()
```

使用 `urllib.request.quote()` 对关键词部分进行编码, 编码之后再构造为完整 URL。随后, 打开对应的 5.html 文件, 发现已经成功爬取了对应内容, 如图 4-16 所示。

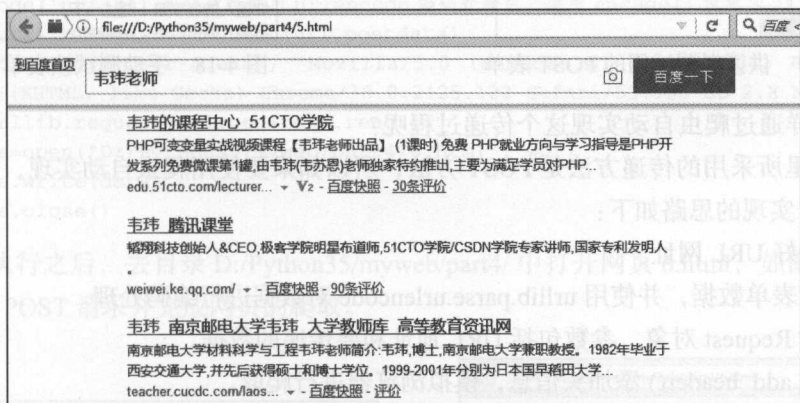


图 4-16 解决中文编码问题

通过以上实例我们可以知道, 如果要使用 GET 请求, 思路如下:

1) 构建对应的 URL 地址, 该 URL 地址包含 GET 请求的字段名和字段内容等信息, 并

且 URL 地址满足 GET 请求的格式, 即“http:// 网址? 字段名 1= 字段内容 1& 字段名 2= 字段内容 2”。

2) 以对应的 URL 为参数, 构建 Request 对象。

3) 通过 `urlopen()` 打开构建的 Request 对象。

4) 按需求进行后续的处理操作, 比如读取网页的内容、将内容写入文件等。

以上, 我们为大家通过实例讲解了爬虫中如何使用 GET 请求, 并总结了爬虫中使用 GET 请求的基本步骤和思路。

## 2. POST 请求实例分析

我们在进行注册、登录等操作的时候, 基本上都会遇到 POST 请求, 接下来我们就为大家通过实例来分析如何通过爬虫来实现 POST 请求。

由于登录需要用到 Cookie 的知识, 所以关于如何登录这一块的内容, 我们将在第 5 章学完了 Cookie 的知识之后, 再为大家详细介绍, 在此, 我们仅需要知道如何使用爬虫通过 POST 表单传递信息即可。

笔者为大家提供了一个 POST 表单的测试网页, 供大家学习时做测试使用, 网址为: <http://www.iqianyue.com/mypost/>。打开对应网址, 发现有一个表单, 如图 4-17 所示。

输入对应数据并单击提交后会发现, 我们提交的信息会使用 POST 方法传递到下方显示, 如图 4-18 所示 (提交的信息是“测试一下”、“weisuen”)。

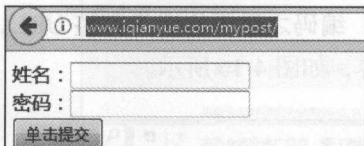


图 4-17 供读者测试用的 POST 表单

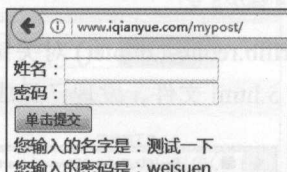


图 4-18 手动测试该表单

那么怎样通过爬虫自动实现这个传递过程呢?

因为这里所采用的传递方法是 POST 方法, 所以如果要使用爬虫自动实现, 我们要构造 POST 请求, 实现的思路如下:

- 1) 设置好 URL 网址。
- 2) 构建表单数据, 并使用 `urllib.parse.urlencode` 对数据进行编码处理。
- 3) 创建 Request 对象, 参数包括 URL 地址和要传递的数据。
- 4) 使用 `add_header()` 添加头信息, 模拟浏览器进行爬取。
- 5) 使用 `urllib.request.urlopen()` 打开对应的 Request 对象, 完成信息的传递。
- 6) 后续处理, 比如读取网页内容、将内容写入文件等。

首先, 需要设置好对应的 URL 地址, 分析该网页, 在单击提交之后, 会传递到当前页面进行处理, 所以处理的页面应该是 <http://www.iqianyue.com/mypost/>, 所以, URL 应该设置



为“http://www.iqianyue.com/mypost/”。

然后我们需要构建表单数据，在该网页上右击“查看页面源代码”，找到对应的 form 表单部分，如图 4-19 所示，然后进行分析。

可以发现，表单中的姓名对应的输入框中，name 属性值为“name”，密码对应的输入框中，name 属性值为“pass”，所以，我们单位构造的数据中会包含两个字段，字段名分别是“name”、“pass”，字段值设置为对应的我们要传递的值。格式为字典形式，即：

```
{ 字段名 1: 字段值 1, 字段名 2: 字段值 2, ...}
```

所以，我们要构造的数据可以为{"name": "ceo@iqianyue.com", "pass": "aA123456"}，将要传递的姓名设置为“ceo@iqianyue.com”，要传递的密码设置为了“aA123456”。设置好数据之后，需要使用 urllib.parse.urlencode 对数据进行编码处理。

然后，我们还需要创建 Request 对象，参数包括 URL 地址和要传递的数据。可以这样创建：req=urllib.request.Request (url 地址，传递的数据)。

接下来，可以使用 add\_header() 添加头信息，模拟浏览器进行爬取，并且使用 urllib.request.urlopen() 打开对应的 Request 对象，完成信息的传递并进行后续处理等，这一部分内容我们之前提过，这里不再赘述。

整合一下上面的内容，该实例中完整的爬虫代码如下所示：

```
import urllib.request
import urllib.parse
url = "http://www.iqianyue.com/mypost/"
postdata =urllib.parse.urlencode({
    "name": "ceo@iqianyue.com",
    "pass": "aA123456"
}).encode('utf-8') # 将数据使用 urlencode 编码处理后，使用 encode() 设置为 utf-8 编码
req = urllib.request.Request(url,postdata)
req.add_header('User-Agent', 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0')
data=urllib.request.urlopen(req).read()
fhandle=open("D:/Python35/myweb/part4/6.html", "wb")
fhandle.write(data)
fhandle.close()
```

按 F5 执行之后，去目录 D:/Python35/myweb/part4/ 中打开网页 6.html，如图 4-20 所示，则成功实现 POST 请求并完成网页的爬取。

```
9 <form action="" method="post">
0 姓名: <input name="name" type="text" /><br>
1 密码: <input name="pass" type="text" /><br>
2 <input name="" type="submit" value="单击提交" />
3 <br />
```

图 4-19 该表单网页的 HTML 代码

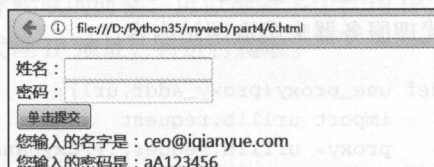


图 4-20 使用爬虫自动提交 POST 表单



## 4.6 代理服务器的设置

有时使用同一个 IP 去爬取同一个网站上的网页，久了之后会被该网站服务器屏蔽。那么怎样解决这个问题呢？

解决思路很简单，即“瞒天过海，暗度陈仓”。

如果我们爬取别人网站的时候，在对方服务器上显示的是别人的 IP 地址，那么，即使对方将显示出来的这个 IP 地址屏蔽了，也无关紧要，因为我们可以换另一个 IP 地址继续爬取。

使用代理服务器，就可以很好地解决这个问题。使用代理服务器去爬取某个网站的内容的时候，在对方网站上，显示的不是我们真实的 IP 地址，而是代理服务器的 IP 地址。并且在 Python 爬虫中，使用代理服务器设置起来也很简单。

那么这些代理服务器可以从哪里找到呢？我们可以在互联网中搜索对应的代理服务器地址，当然也可以从整理好的网址 <http://yum.iqianyue.com/proxy> 中找到很多代理服务器地址。打开网页，可以找到类似如图 4-21 所示的界面。

国内高匿代理IP							更多
国家	代理IP地址	端口	服务器地址	是否匿名	类型	存活时间	验证时间
中国	113.3.78.124	8118	黑龙江	高匿	HTTP	2天	不到1分钟
中国	202.75.210.45	7777	浙江杭州	高匿	HTTP	59天	不到1分钟
中国	111.155.124.83	8123	北京	高匿	HTTP	5天	不到1分钟
中国	218.30.99.209	81	北京	高匿	HTTP	45天	1分钟前
中国	210.72.14.142	80	上海	高匿	HTTP	65天	2分钟前
中国	183.61.71.112	8888	广东广州	高匿	HTTP	111天	3分钟前
中国	61.227.228.141	8080	台湾台北	高匿	HTTP	3小时	4分钟前
中国	61.224.239.71	8080	台湾台北	高匿	HTTP	1小时	4分钟前
中国	171.38.85.73	8123	广西玉林	高匿	HTTP	290天	4分钟前
中国	171.38.196.16	8123	广西钦州	高匿	HTTP	49天	4分钟前
中国	122.114.36.13	808	河南郑州	高匿	HTTP	1天	5分钟前
中国	180.103.131.65	808	江苏淮安	高匿	HTTP	2天	5分钟前

图 4-21 最新代理服务器列表

可以看到，这里会更新很多代理 IP 地址，我们尽量找验证时间比较短的，这些成功的概率会比较大，一些验证时间较长的，可能会失效。

此时，我们可以选择第二个代理 IP 地址 202.75.210.45，对应的端口号是 7777，完整的格式为：“网址:端口号”，即 202.75.210.45:7777。

用了代理 IP 地址之后，我们就可以进行相应程序的编写了、可以使用以下程序，实现通过代理服务器来爬取网站内容。

```
def use_proxy(proxy_addr,url):
    import urllib.request
    proxy= urllib.request.ProxyHandler({'http':proxy_addr})
    opener = urllib.request.build_opener(proxy, urllib.request.HTTPHandler)
    urllib.request.install_opener(opener)
```

```
data = urllib.request.urlopen(url).read().decode('utf-8')
return data
proxy_addr="202.75.210.45:7777"
data=use_proxy(proxy_addr,"http://www.baidu.com")
print(len(data))
```

我们首先建立一个名为 `use_proxy` 的自定义函数，该函数主要实现使用代理服务器来爬去某个 URL 网页的功能。

在函数中，我们设置两个形参，第一个形参为代理服务器的地址，第二个形参代表要爬取的网页的地址。

然后，使用 `urllib.request.ProxyHandler()` 来设置对应的代理服务器信息，设置格式为：`urllib.request.ProxyHandler({ 'http': 代理服务器地址 })`，接下来，使用 `urllib.request.build_opener()` 创建了一个自定义的 `opener` 对象，第一个参数为代理信息，第二个参数为 `urllib.request.HTTPHandler` 类。

为了方便，可以使用 `urllib.request.install_opener()` 创建全局默认的 `opener` 对象，那么，在使用 `urlopen()` 时亦会使用我们安装的 `opener` 对象，所以我们下面才可以直接使用 `urllib.request.urlopen()` 打开对应网址爬取网页并读取，编码后赋给变量 `data`，最后返回 `data` 的值给函数。

随后，在函数外设置好对应的代理 IP 地址，然后调用自定义函数 `use_proxy`，并传递两个实参，分别为使用的代理地址及要爬取的网址。将函数的调用结果赋值给变量 `data`，并输出 `data` 内容的长度。当然，也可以将 `data` 的值写进某个文件当中存储起来。

以上程序执行结果如下：

```
===== RESTART: D:\Python35\4.6.py =====
99298
```

此时，可以看到成功使用代理服务器爬取到了百度首页，并返回获取内容的大小。如果此时代理服务器地址失效或填写错了代理服务器，则会发生错误，比如，我们故意将代理服务器改一个数，改为“202.75.210.46:7777”，此时出现的结果为：

```
===== RESTART: D:\Python35\4.6.py =====
urllib.error.URLError: <urlopen error [WinError 10060] 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。>
```

所以，如果在使用代理服务器进行网站爬取时，出现相应异常，则需要考虑是否对应的代理 IP 失效了，如果是，换一个代理 IP 即可。实际爬取的时候，可以准备多个代理 IP，轮流进行爬取，若失效，则程序直接自动替换为其他代理 IP 地址，再进行爬取。

## 4.7 DebugLog 实战

有的时候，我们希望在程序运行的过程中，边运行边打印调试日志，此时需要开启

DebugLog。

如何开启 DebugLog 呢？思路如下：

1) 分别使用 `urllib.request.HTTPHandler()` 和 `urllib.request.HTTPSHandler()` 将 `debuglevel` 设置为 1。

2) 使用 `urllib.request.build_opener()` 创建自定义的 `opener` 对象，并使用 1) 中设置的值作为参数。

3) 用 `urllib.request.install_opener()` 创建全局默认的 `opener` 对象，这样，在使用 `urlopen()` 时，也会使用我们安装的 `opener` 对象。

4) 进行后续相应的操作，比如 `urlopen()` 等。

此时，根据以上思路，我们可以通过如下代码开启 DebugLog：

```
import urllib.request
httphd=urllib.request.HTTPHandler(debuglevel=1)
httpshd=urllib.request.HTTPSHandler(debuglevel=1)
opener=urllib.request.build_opener(httphd,httpshd)
urllib.request.install_opener(opener)
data=urllib.request.urlopen("http://edu.51cto.com")
```

上述程序中，我们开启了 DebugLog，并使用 `urllib.request.urlopen()` 爬取 51CTO 学院首页的信息。执行结果如下：

```
===== RESTART: D:\Python35\4.7.py =====
send: b'GET / HTTP/1.1\r\nAccept-Encoding: identity\r\nUser-Agent: Python-urllib/3.5\r\nConnection: close\r\nHost: edu.51cto.com\r\n\r\n'
reply: 'HTTP/1.1 200 OK\r\n'
header: Date header: Content-Type header: Transfer-Encoding header: Connection
header: Set-Cookie header: Server header: Vary header: Set-Cookie header:
Expires header: Cache-Control header: Pragma header: Load-Balancing
```

可以看到，此时会边执行程序，边打印调试的 Log 日志，成功开启 DebugLog。

## 4.8 异常处理神器——URLError 实战

程序在执行的过程中，难免会发生异常，发生异常不要紧，关键是要能合理地处理异常，在 Python 爬虫中，经常要处理一些与 URL 相关的异常。此时，我们可以使用 URL 异常处理神器——URLError 类进行相应的处理，使用 URLError 类，我们首先要导入 `urllib.error` 模块。

进行异常处理，我们经常使用 `try...except` 语句，在 `try` 中执行主要代码，在 `except` 中捕获异常信息，并进行相应的异常处理。

这一节我们主要介绍两个类，第一个类是 URLError 类，第二个类是 URLError 类的一个子类——HTTPError 类。

首先,我们来看第一个实例(实例1):

```
import urllib.request
import urllib.error
try:
    urllib.request.urlopen("http://blog.csdn.net")
except urllib.error.URLError as e:
    print(e.code)
    print(e.reason)
```

在该实例中,我们要对网址“http://blog.csdn.net”进行爬取,CSDN 博客是禁止对文章进行爬取的,而在这里,没有模拟浏览器去爬取,所以必然会出现 403 的错误,此时会引发 except 部分,并通过 urllib.error.URLError as e 捕获异常信息 e,然后进行相应的异常处理。这里的异常处理会输出对应的异常状态和异常原因。

执行该实例,会产生以下结果:

```
===== RESTART: D:\Python35\4.8.py =====
403
Forbidden
```

我们可以看到,其输出了对应的异常状态码和异常原因。

一般来说,产生 URLError 的原因有如下几种可能:

- 1) 连接不上服务器
- 2) 远程 URL 不存在
- 3) 无网络
- 4) 触发了 HTTPError

显然,刚才产生 URLError 异常不属于前三种情况,而是由于触发了 HTTPError。所以,以上代码异常处理我们可以直接用 HTTPError 代替 URLError。

可以将代码写成:

```
import urllib.request
import urllib.error
try:
    urllib.request.urlopen("http://blog.csdn.net")
except urllib.error.HTTPError as e:
    print(e.code)
    print(e.reason)
```

可以看到,此时我们直接使用子类 HTTPError 进行异常处理,同样可以输出如下结果:

```
===== RESTART: D:\Python35\4.8.py =====
403
Forbidden
```

这里的状态码为 403,这个状态码是服务器返回给客户端的一个编码,不同的编码具有



不同的含义。

接下来我们总结一下常见的状态码及含义：

200	OK
	一切正常
301	Moved Permanently
	重定向到新的 URL，永久性
302	Found
	重定向到临时的 URL，非永久性
304	Not Modified
	请求的资源未更新
400	Bad Request
	非法请求
401	Unauthorized
	请求未经授权
403	Forbidden
	禁止访问
404	Not Found
	没有找到对应页面
500	Internal Server Error
	服务器内部出现错误
501	Not Implemented
	服务器不支持实现请求所需要的功能

以上我们总结了常见的状态码以及其对应含义，以后在遇到这些状态码的时候，就知道是什么原因所导致的异常了。

HTTPError 子类无法处理刚才所提到的产生 URLError 的前 3 种原因的异常，即无法处理：连接不上服务器、远程 URL 不存在、无网络引起的异常。

比如，我们构造一个不存在的网址，引发远程 URL 不存在的异常，此时，不能够通过 HTTPError 处理，但能通过 URLError 处理。

我们写了以下代码进行实验：

```
import urllib.request
import urllib.error
try:
    urllib.request.urlopen("http://blog.baidussss.net")
except urllib.error.HTTPError as e:
    print(e.reason)
```

执行结果如下：

```
===== RESTART: D:\Python35\4.8.py =====
Traceback (most recent call last):
  File "D:\Python35\lib\urllib\request.py", line 1254, in do_open
    h.request(req.get_method(), req.selector, req.data, headers)
  File "D:\Python35\lib\http\client.py", line 1106, in request
    self._send_request(method, url, body, headers)
.....
```



可以发现,此时无法进行异常处理,但是此时,如果使用 URLError,是可以进行异常处理的,比如,将代码改写成如下代码即可正确执行:

```
import urllib.request
import urllib.error
try:
    urllib.request.urlopen("http://blog.baidusss.net")
except urllib.error.URLError as e:
    print(e.reason)
```

执行结果如下:

```
===== RESTART: D:\Python35\4.8.py =====
[Errno 11004] getaddrinfo failed
```

可以看到,成功进行了异常处理,同时输出了异常原因。

在实际处理异常时,我们并不知道使用 HTTPError 能不能处理。如果,异常处理中只有 HTTPError 子类的话,若发生连接不上服务器、远程 URL 不存在、无网络等异常,是无法处理的。所以,我们可以对以上代码进行相应的优化,先让其用 HTTPError 子类进行处理,若无法处理,再让其用 URLError 进行处理,优化后的代码如下所示:

```
import urllib.request
import urllib.error
try:
    urllib.request.urlopen("http://blog.baidusss.net")
except urllib.error.HTTPError as e:
    print(e.code)
    print(e.reason)
except urllib.error.URLError as e:
    print(e.reason)
```

在上述代码中,我们先利用子类进行异常处理,若无法处理,再用父类进行异常处理,此时,不管发生的是哪种异常,都能够进行完美处理。上述代码的执行结果如下:

```
===== RESTART: D:\Python35\4.8.py =====
[Errno 11004] getaddrinfo failed
```

可以看到,此时用子类无法处理,那么将自动交给父类进行相应处理。

思考:

1) URLError 是 HTTPError 的父类,那么我们能不能用 URLError 直接代替 HTTPError 呢?

2) 以上程序中分别用了 HTTPError 和 URLError 进行处理,我们能否整合一下,用其中的一个类就能处理完呢?如果能,应该怎么改进?

首先,有时不能直接用 URLError 代替 HTTPError,但改进后可以整合。为什么有时不能直接代替呢?

我们知道,在本节的实例1中,用 `URLError` 代替 `HTTPError` 可以直接运行,我们将本节实例1中的代码拿过来,但是改一下需要打开的URL:

```
import urllib.request
import urllib.error
try:
    urllib.request.urlopen("http://www.baidussssss.net")
except urllib.error.URLError as e:
    print(e.code)
    print(e.reason)
```

可以发现,这个代码执行起来是有问题的,如下所示:

```
===== RESTART: D:\Python35\4.8.py =====
Traceback (most recent call last):
  File "D:\Python35\lib\urllib\request.py", line 1254, in do_open
    h.request(req.get_method(), req.selector, req.data, headers)
  File "D:\Python35\lib\http\client.py", line 1106, in request
    self._send_request(method, url, body, headers)
.....
```

因为此时触发的原因是连接不上服务器、远程URL不存在、无网络等异常中的一个,即此时没有 `e.code`,只有 `e.reason`,所以自然无法输出 `e.code`,虽然我们把 `e.code` 部分去掉就可以解决异常问题,但如果到时引发的是 `HTTPError`,我们又希望获取对应的状态码怎么办?去掉了 `e.code` 就无法获得状态码了,即“顾此失彼”。

那么,如果我们要整合,怎么平衡一下呢?

可以这样做,我们使用 `URLError` 进行异常处理,但是做一个判断,如果有 `e.code` 则输出对应信息,如果没有则自动忽略。同理,如果有 `e.reason` 则输出 `e.reason`,没有亦忽略该项。可以使用 `hasattr()` 函数来判断是否有这些属性。加上这个判断之后,不管是何种原因引发的URL异常,都能够通过 `URLError` 进行处理,代码改进如下:

```
# 整合后代码
import urllib.request
import urllib.error
try:
    urllib.request.urlopen("http://blog.csdn.net")
except urllib.error.URLError as e:
    if hasattr(e, "code"):
        print(e.code)
    if hasattr(e, "reason"):
        print(e.reason)
```

此时,如果是引发了 `HTTPError` 异常,则判断出有 `e.code`,就会既输出状态码也输出错误原因,若引发异常的原因是连接不上服务器、远程URL不存在、无网络等异常中的一个,则判断没有 `e.code`,所以此时只输出 `e.reason`,不管是何种原因,都能解决。

通过前面的学习,相信大家对 `URLError` 异常处理已经有了较为深入的理解。

## 4.9 小结

1) Urllib 是 Python 提供的一个用于操作 URL 的模块, 在 Python2.X 中, 有 Urllib 库, 也有 Urllib2 库, 在 Python3.X 中 urllib2 合并到了 urllib 中, 我们爬取网页的时候, 经常需要用到这个库。

2) 一般来说, URL 标准中只会允许一部分 ASCII 字符, 比如数字、字母、部分符号等, 而其他的一些字符, 比如汉字等, 是不符合 URL 标准的。所以如果我们在 URL 中使用不符合标准的字符就会出现异常, 此时需要进行 URL 编码方可解决。比如在 URL 中输入中文或者“:”或者“&”等不符合标准的字符时, 需要编码。

3) 当我们无法爬取一些网页时可能会出现 403 错误, 因为这些网页为了防止别人恶意采集其信息进行了一些反爬虫的设置。

4) 由于 `urlopen()` 不支持一些 HTTP 的高级功能, 所以, 我们如果要修改报头, 可以使用 `urllib.request.build_opener()` 进行。

5) 我们还可以使用 `urllib.request.Request()` 下的 `add_header()` 实现浏览器模拟技术。

6) 程序在执行的过程中, 难免会发生异常, 发生异常不要紧, 关键是要能合理地处理异常, 在 Python 爬虫中, 经常要处理一些与 URL 相关的异常。此时, 我们可以使用 URL 异常处理神器——`URLError` 类进行相应的处理。

7) 我们经常使用 `try...except` 语句进行异常处理, 在 `try` 中执行主要代码, 在 `except` 中捕获异常信息, 并进行相应的异常处理。

## Chapter 3 第 5 章

# 正则表达式与 Cookie 的使用

有时我们在进行字符串处理的时候，希望按自定义的规则进行处理，我们将这些规则称为模式。我们可以用正则表达式来描述这些自定义规则，正则表达式也称为模式表达式。

## 5.1 什么是正则表达式

那么什么是正则表达式呢？

简单来说，正则表达式就是描述字符串排列的一套规则。比如，我们想找出一个网页中的所有电子邮件，其他的信息需要过滤掉，那么此时，我们可以观察电子邮件的格式，然后，写一个正则表达式来表示所有的电子邮件，随后，我们可以利用该正则表达式从网页中提取出所有满足该规则的字符串出来。这些满足规则的字符串，必然是电子邮件格式，故而此时，我们可以轻松地将该网页中的所有电子邮件提取出来，并过滤掉其他信息。

利用正则表达式，还可以做很多事情，其主要用于字符串的匹配。在实际项目中，我们经常需要找到某一类符合某种格式的信息，此时，我们可以观察这些数据的规律，然后将这些数据的格式规律用正则表达式描述出来，然后利用正则表达式函数进行相应的处理即可。

可以看到，正则表达式的功能是非常强大的，并且在实际项目中，对于要处理特定格式的信息，经常要用到正则表达式。

在 Python 中，一般我们会使用 re 模块实现 Python 正则表达式的功能。

## 5.2 正则表达式基础知识

接下来，我们跟大家一起学习一下正则表达式的基础知识。

本节中我们主要为大家介绍正则表达式如何去写，我们将分别从原子、元字符、模式修正符、贪婪模式与懒惰模式等方面介绍。关于正则表达式的使用则会在下一节中详细介绍。

## 1. 原子

原子是正则表达式中最基本的组成单位，每个正则表达式中至少要包含一个原子，常见的原子有以下几类：

- 1) 普通字符作为原子。
- 2) 非打印字符作为原子。
- 3) 通用字符作为原子。
- 4) 原子表。

接下来，我们为大家分别讲解。

### (1) 普通字符作为原子

首先介绍普通字符作为原子的情况。我们可以使用一些普通的字符，比如数字、大小写字母、下划线等都可以作为原子使用。比如如下程序中，我们就使用了“yue”作为原子使用，这里有3个原子，分别是“y”“u”“e”。

```
import re
pattern="yue" # 普通字符作为原子
string="http://yum.iqianyue.com"
result1=re.search(pattern,string)
print(result1)
```

在上面的程序中，我们要使用正则表达式，所以导入了re模块。导入了re模块之后，我们设置了正则表达式，并将正则表达式的值赋给了变量pattern，随后，我们定义了一串字符串，然后使用re模块里面的search()函数，从字符串string中去匹配对应的正则表达式，若匹配成功，则将匹配结果返回给变量result1，并打印出来。此时执行结果如下所示，可以看到，其成功匹配到了对应的结果“yue”，因为在字符串string中包含对应的符合正则表达式规则的字符串。

```
>>>
===== RESTART: D:\Python35\5.1.py =====
<_sre.SRE_Match object; span=(16, 19), match='yue'>
```

### (2) 非打印字符作为原子

接下来，讲解非打印字符作为原子的情况。所谓的非打印字符，指的是有一些在字符串中用于格式控制的符号，比如换行符等。在此只为大家讲解常用的一些非打印字符，如表5-1所示：

比如，可以输入以下程序来实现换行符的匹配。

```
import re
pattern="\n"
```

表 5-1 常用的非打印字符

符 号	含 义
\n	用于匹配一个换行符
\t	用于匹配一个制表符



```
string='''http://yum.iqianyue.com
http://baidu.com'''
result1=re.search(pattern,string)
print(result1)
```

可以看到，程序中的字符串变量 `string` 包含一个多行的数据，所以这个数据里面包含了对应的换行符，所以，此时进行换行符匹配，就能够匹配成功。此时的执行结果如下所示：

```
>>>
===== RESTART: D:\Python35\5.1.py =====
<_sre.SRE_Match object; span=(23, 24), match='\n'>
```

可以看到，只是成功的匹配了一个换行符 `\n`，如果我们将程序改一下，改为如下程序：

```
import re
pattern="\n"
string='''http://yum.iqianyue.comhttp://baidu.com'''
result1=re.search(pattern,string)
print(result1)
```

由于在源字符串 `string` 中不包含换行，所以无法成功匹配，执行结果如下所示，我们可以看到输出了 `None`，即没有匹配结果。

```
>>>
===== RESTART: D:\Python35\5.1.py =====
None
```

### （3）通用字符作为原子

再接下来，讲解通用字符作为原子的情况，所谓的通用字符，即一个原子可以匹配一类字符，我们在实际项目中经常会用到这一类原子。常见的通用字符及其含义如表 5-2 所示：

表 5-2 常见的通用字符及其含义

符 号	含 义
<code>\w</code>	匹配任意一个字母、数字或下划线
<code>\W</code>	匹配除字母、数字和下划线以外的任意一个字符
<code>\d</code>	匹配任意一个十进制数
<code>\D</code>	匹配除十进制数以外的任意一个其他字符
<code>\s</code>	匹配任意一个空白字符
<code>\S</code>	匹配除空白字符以外的任意一个其他字符

比如，我们可以使用 `"\w\dpython\w"` 对 `"python"` 字符进行匹配，字符后是一个字母、数字或下划线，字符前一位是一个任意的十进制数，再前一位是一个字母数字或下划线的格式的字符串，如 `"67python8"`、`"u2python_"` 但均可以匹配成功，因为这些字符都符合我们所设置的格式。我们可以输入以下程序并运行：

```
import re
pattern="\w\dpython\w"
string="abcdfphp345pythony_py"
result1=re.search(pattern,string)
print(result1)
```

执行结果如下所示:

```
>>>
===== RESTART: D:\Python35\5.1.py =====
<_sre.SRE_Match object; span=(9, 18), match='45pythony'>
```

我们可以看得到,此时可以成功匹配,匹配的结果为'45pythony',因为其符合我们所设置的格式,我们在进行对应信息提取的时候,也经常要观察这些信息的规律,然后使用正则表达式描述出对应信息的格式,这样就可以轻松地将某一类信息提取出来。

#### (4) 原子表

接下来讲解原子表的使用。使用原子表,可以定义一组地位平等的原子,然后匹配的时候会取该原子表中的任意一个原子进行匹配,在 Python 中,原子表由 [] 表示,比如 [xyz] 就是一个原子表,这个原子表中定义了 3 个原子,这 3 个原子的地位平等,如,我们定义的正则表达式为 "[xyz]py",对应的源字符串是 "xpython",如果此时使用 re.search() 函数进行匹配,就可以匹配出结果 "xpy",因为此时只要 py 前一位是 x y z 字母中的其中一个字母,就可以匹配成功。

类似的,[^]代表的是除了中括号里面的原子均可以匹配,比如 "[^xyz]py" 能匹配 "apy",但是却不能匹配 "xpy" 等。

我们可以输入如下程序:

```
import re
pattern1="\w\dpython[xyz]\w"
pattern2="\w\dpython[^xyz]\w"
pattern3="\w\dpython[xyz]\W"
string="abcdfphp345pythony_py"
result1=re.search(pattern1,string)
result2=re.search(pattern2,string)
result3=re.search(pattern3,string)
print(result1)
print(result2)
print(result3)
```

该程序的执行结果如下:

```
>>>
===== RESTART: D:\Python35\5.1.py =====
<_sre.SRE_Match object; span=(9, 19), match='45pythony_'>
None
None
```

我们发现，程序中的 `result1` 匹配成功，匹配结果是“45pythony”，而 `result2` 和 `result3` 则没有匹配成功。因为源字符串 `string` 中含有符合 `pattern1` 格式的字符串，而没有符合 `pattern2` 或 `pattern3` 格式的字符串。

## 2. 元字符

以上我们讲解了一些常见原子的使用，接下来我们为大家讲解元字符的使用。

所谓的元字符，就是正则表达式中具有一些特殊含义的字符，比如重复  $N$  次前面的字符等。

在此，笔者为大家整理了一些常见的元字符，如表 5-3 所示：

表 5-3 常见的元字符及其含义

符 号	含 义
.	匹配除换行符以外的任意字符
^	匹配字符串的开始位置
\$	匹配字符串的结束位置
*	匹配 0 次、1 次或多次前面的原子
?	匹配 0 次或 1 次前面的原子
+	匹配 1 次或多次前面的原子
{n}	前面的原子恰好出现 $n$ 次
{n,}	前面的原子至少出现 $n$ 次
{n, m}	前面的原子至少出现 $n$ 次，至多出现 $m$ 次
	模式选择符
()	模式单元符

具体来说，元字符可以分为：任意匹配元字符、边界限制元字符、限定符、模式选择符、模式单元等。

### (1) 任意匹配元字符

首先讲解任意匹配元字符“.”，我们可以使用“.”匹配一个除换行符以外的任意字符。比如，我们可以使用正则表达式“`.python...`”匹配一个“python”字符前面有 1 位，后面有 3 位格式的字符，这前面的 1 位和后面的 3 位可以是除了换行符以外的任意字符。

可以输入如下程序：

```
import re
pattern=".python..."
string="abdcdfphp345pythony_py"
result1=re.search(pattern,string)
print(result1)
```

执行结果如下：

```
>>>
===== RESTART: D:\Python35\5.1.py =====
<_sre.SRE_Match object; span=(10, 20), match='5pythony_p'>
```

可以看到, 该正则表达式成功从源字符串 `string` 中匹配到了字符串 `'5pythony_p'`。

## (2) 边界限制元字符

接下来, 讲解边界限制符, 可以使用 `“^”` 匹配字符串的开始, 使用 `“$”` 匹配字符串的结束。我们通过以下实例进行分析:

```
import re
pattern1="^abd"
pattern2="^abc"
pattern3="py$"
pattern4="ay$"
string="abcdfphp345pythony_py"
result1=re.search(pattern1,string)
result2=re.search(pattern2,string)
result3=re.search(pattern3,string)
result4=re.search(pattern4,string)
print(result1)
print(result2)
print(result3)
print(result4)
```

执行结果如下:

```
>>>
===== RESTART: D:\Python35\5.1.py =====
None
<_sre.SRE_Match object; span=(0, 3), match='abc'>
<_sre.SRE_Match object; span=(19, 21), match='py'>
None
```

可以看到, 第一行和第四行没有匹配的结果, 而第二和第三行这可以成功匹配。以上程序中, 正则表达式 `pattern1` 限制了必须要以 `“abd”` 开头, 如果不以 `“abd”` 开头, 则无法成功匹配, 本例中源字符串 `string` 为 `"abcdfphp345pythony_py"`, 显然不符合正则表达式格式, 故而无法匹配; 而正则表达式 `pattern2` 限制了必须要以 `“abc”` 开头, 此时原字符串中有符合条件的结果, 故而成功匹配; 正则表达式 `pattern3` 限制了必须要以 `“py”` 结尾, 源字符串中有符合条件的结果; 而 `pattern4` 限制了必须要以 `“ay”` 结尾, 此时源字符串中没有符合条件的结果。

## (3) 限定符

接下来讲解限定符的使用, 限定符也是元字符中的一种, 常见的限定符包括 `*`、`?`、`+`、`{n}`、`{n,}`、`{n, m}`, 具体的含义我们可以参照上面的表格。接下来我们通过一个实例来分析一下限定符的使用。

```
import re
pattern1="py.*n"
pattern2="cd{2}"
```



```

pattern3="cd{3}"
pattern4="cd{2,}"
string="abcdcdfphp345pythony_py"
result1=re.search(pattern1,string)
result2=re.search(pattern2,string)
result3=re.search(pattern3,string)
result4=re.search(pattern4,string)
print(result1)
print(result2)
print(result3)
print(result4)

```

该程序的执行结果如下：

```

>>>
===== RESTART: D:\Python35\5.1.py =====
<_sre.SRE_Match object; span=(13, 19), match='python'>
<_sre.SRE_Match object; span=(2, 5), match='cdd'>
<_sre.SRE_Match object; span=(2, 6), match='cddd'>
<_sre.SRE_Match object; span=(2, 6), match='cddd'>

```

可以看到，4个正则表达式均成功进行了匹配，但匹配结果有所不同。正则表达式 `pattern1` 中，我们设置的格式是“py”与“n”之间可以是除换行符以外的任意字符，并且该任意字符可以出现0次、1次或多次，所以此时，匹配出了结果‘python’；而正则表达式 `pattern2` 中要求“cd”字符串中的“d”恰好出现两次，所以此时匹配出了结果‘cdd’；正则表达式 `pattern3` 中要求“cd”字符串中的“d”恰好出现3次，所以此时匹配出的结果为“cddd”；而在正则表达式 `pattern4` 中，要求“cd”字符串中的“d”至少出现两次，所以此时只要满足格式，就会在源字符串中尽可能多地匹配字符，所以匹配到的结果是“cddd”。这里，我们通过实例为大家分析了一些限定符的使用。

#### （4）模式选择符

接下来，讲解模式选择符“|”，使用模式选择符，可以设置多个模式，匹配时，可以从中选择任意一个模式匹配。比如正则表达式“python|php”中，字符串“python”和“php”均满足匹配条件。接下来我们通过实例讲解：

```

import re
pattern="python|php"
string="abcdcfphp345pythony_py"
result1=re.search(pattern,string)
print(result1)

```

该程序的执行结果如下：

```

>>>
===== RESTART: D:\Python35\5.1.py =====
<_sre.SRE_Match object; span=(5, 8), match='php'>

```

可以看到，该正则表达式成功从源字符串中匹配到了结果“php”。

### (5) 模式单元符

接下来我们为大家讲解元字符中的模式单元符“()”，可以使用“()”将一些原子组合成一个大原子使用，小括号括起来的部分会被当做一个整体去使用。

比如，我们可以输入以下程序并执行：

```
import re
pattern1="(cd){1,}"
pattern2="cd{1,}"
string="abcdcdcdcdfphp345pythony_py"
result1=re.search(pattern1,string)
result2=re.search(pattern2,string)
print(result1)
print(result2)
```

执行结果如下所示：

```
>>>
===== RESTART: D:\Python35\5.1.py =====
<_sre.SRE_Match object; span=(2, 10), match='cdcdcdcd'>
<_sre.SRE_Match object; span=(2, 4), match='cd'>
```

可以看到，正则表达式 pattern1 中，cd 被看成一个大原子，此时的含义代表“cd”整体至少出现一次，当然会尽量多的匹配，所以此时，可以从源字符串 "abcdcdcdcdfphp345pythony\_py" 中匹配出结果 'cdcdcdcd'；而在正则表达式 pattern2 中，其含义是 d 原子至少出现 1 次，而不会把 cd 看为一个整体，所以此时只能匹配到结果 'cd'。

## 3. 模式修正

上面讲解了一些常见的元字符的使用，接下来讲解模式修正符、所谓模式修正符，即可以在不改变正则表达式的情况下，通过模式修正符改变正则表达式的含义，从而实现一些匹配结果的调整等功能。比如，可以使用模式修正符 I 让对于模式在匹配时不区分大小写。

常见的一些模式修正符如表 5-4 所示：

表 5-4 常见的模式修正符及其含义

符 号	含 义
I	匹配时忽略大小写
M	多行匹配
L	做本地化识别匹配
U	根据 Unicode 字符及解析字符
S	让 . 匹配包括换行符，即用了该模式修正后，“.” 匹配就可以匹配任意的字符了

接下来我们通过一个实例来分析：

```
import re
pattern1="python"
pattern2="python"
string="abcdfphp345Pythony_py"
result1=re.search(pattern1,string)
result2=re.search(pattern2,string,re.I)
print(result1)
print(result2)
```

程序的执行结果如下：

```
>>>
===== RESTART: D:\Python35\5.1.py =====
None
<_sre.SRE_Match object; span=(11, 17), match='Python'>
```

可以看到，第一行没有匹配到任何结果，因为此时匹配是区分大小写的，即正则表达式与源字符串中的内容大小写要完全一致才能够匹配到。而第二行这匹配出了结果“Python”，因为，在执行 `re.research()` 函数的时候，通过第三个参数设置了模式修正 `re.I`，即让其在匹配时忽略大小写，故而只要内容一样，即使大小写不一样也可以匹配成功。

至此，我们讲解了正则表达式的基本使用方法，接下来我们再为大家扩展一个知识点：贪婪模式和懒惰模式。

#### 4. 贪婪模式与懒惰模式

• 总的来说，贪婪模式的核心点就是尽可能多地匹配，而懒惰模式的核心点就是尽可能少地匹配。通过下面一个实例我们可以更好的理解这句话：

```
import re
pattern1="p.*y"# 贪婪模式
pattern2="p.*?y"# 懒惰模式
string="abcdfphp345pythony_py"
result1=re.search(pattern1,string)
result2=re.search(pattern2,string)
print(result1)
print(result2)
```

执行结果如下：

```
>>>
===== RESTART: D:\Python35\5.1.py =====
<_sre.SRE_Match object; span=(5, 21), match='php345pythony_py'>
<_sre.SRE_Match object; span=(5, 13), match='php345py'>
```

可以看到，通过贪婪模式所匹配出来的结果为 `'php345pythony_py'`，而通过懒惰模式匹配出来的结果为 `'php345py'`，换一句话来说，懒惰模式采用的是就近匹配原则，可以让匹配结果更为精确。上面程序通过贪婪模式匹配，已经找到了一个结尾 `y` 字符了，但仍然不会停

止搜索，直到找不到结尾字符 *y* 为止才停止搜索，所以此时，结尾的字符 *y* 为源字符串中最右边的这个 *y* 字符。而如果使用懒惰模式，一旦搜索到了结尾字符 *y*，就立即停止，所以此时匹配截止到的是距离第 1 个 *p* 字符最近的这个 *y* 字符。

那么怎么设置贪婪模式和懒惰模式呢？

通常情况下，如果我们想在某些字符间匹配任意字符，像 “*p.\*y*” 这样写没有任何的语法错误，这个时候默认是使用贪婪模式的，如果要转化为懒惰模式，需要在对应的 “*.\**” 后面加上 “*?*”，方可转化为懒惰模式。

## 5.3 正则表达式常见函数

通过上一节，我们已经学会了如何写正则表达式，但是这些正则表达式要进行相应匹配还需要通过正则表达式函数来完成。

常见的正则表达式函数有 `re.match()` 函数、`re.search()` 函数、全局匹配函数、`re.sub()` 函数，接下来我们为大家进行分别讲解。

### 1. `re.match()` 函数

如果想要从源字符串的起始位置匹配一个模式，我们可以使用 `re.match()` 函数，`re.match()` 函数的使用格式是：

```
re.match(pattern, string, flag)
```

第一个参数代表对应的正确表达式，第二个参数代表对应的源字符，第三个参数是可选参数，代表对应的标志位，可以放模式修正符等信息。

接下来我们通过实例来进行实际应用：

```
import re
string="apythonhellomypythonhispythonourpythonend"
pattern=".python."
result=re.match(pattern,string)
result2=re.match(pattern,string).span()
print(result)
print(result2)
```

程序执行结果如下：

```
>>>
===== RESTART: D:\Python35\5.2.py =====
<_sre.SRE_Match object; span=(0, 8), match='apythonh'>
(0, 8)
```

程序中，会从 `string` 的起始位置进行匹配，如果不满足模式，则会返回 `None`，如果符合要求，则返回匹配成功的结果。此时，正则表达式刚好可以从 `string` 的开头进行匹配并



匹配成功，所以，可以看到，`result` 和 `result2` 都成功进行了匹配，但是展现形式不一样。通过 `.span()` 设置可以过滤掉一些信息，只留下匹配成功的结果在源字符串中的位置。

## 2. re.search() 函数

类似的，我们还可以使用 `re.search()` 函数进行匹配，使用该函数进行匹配，会扫描整个字符串并进行对应的匹配。该函数与 `re.match()` 函数最大的不同是，`re.match()` 函数从源字符串的开头进行匹配，而 `re.search()` 函数会在全文中进行检索并匹配。

以下实例可以进行更好的说明：

```
import re
string="hellomypythonhispythonourpythonend"
pattern=".python."
result=re.match(pattern,string)
result2=re.search(pattern,string)
print(result)
print(result2)
```

程序的执行结果如下：

```
>>>
===== RESTART: D:\Python35\5.2.py =====
None
<_sre.SRE_Match object; span=(6, 14), match='ypythonh'>
```

可以看到，第一行使用 `re.match()` 函数进行匹配，第二行使用 `re.search()` 函数进行匹配。显然，源字符串 `string` 的开始位置不符合正则表达式格式，所以使用 `re.match()` 匹配不到结果，但是源字符串 `string` 的内容中含有符合正则表达式格式的内容，所以使用 `re.search()` 可以匹配到对应结果 `'ypythonh'`。理解这个例子，可以很好的区分 `re.match()` 函数和 `re.search()` 函数的区别。

## 3. 全局匹配函数

通过观察，可以发现，在以上的匹配中，即便源字符串中有多个结果符合模式，也只会匹配一个结果，那么，我们如何将符合模式的内容全部都匹配出来呢？

思路如下：

- 1) 使用 `re.compile()` 对正则表达式进行预编译。
- 2) 编译后，使用 `findall()` 根据正则表达式从源字符串中将匹配的结果全部找出。

我们可以通过以下实例更好地理解：

```
import re
string="hellomypythonhispythonourpythonend"
pattern=re.compile(".python.")# 预编译
result=pattern.findall(string)# 找出符合模式的所有结果
print(result)
```

执行结果如下：

```
>>>
===== RESTART: D:\Python35\5.2.py =====
['ypythonh', 'spythono', 'rpythone']
```

可以看到，这段代码会将 string 中满足 pattern 模式的结果全部输出，符合条件的结果一共有 3 个：'ypythonh', 'spythono', 'rpythone'。

在上面的实例中，我们首先对正则表达式 ".python." 进行了预编译，并将预编译结果赋给了变量 pattern，然后使用 pattern.findall(string) 将对应符合模式的结果全部找出。

以上过程，分了两步去实现，我们可以将其整合一下，使用 re.compile(pattern).findall(string) 直接实现根据正则表达式从源字符串 string 中找出所有符合模式的结果。

所以，上面的实例可以整合并改写为：

```
import re
string="hellomypythonhispythonourpythonend"
pattern=".python."
result=re.compile(pattern).findall(string)
print(result)
```

此时输出的结果是一样的，只不过将两步的代码合并在一起写了。输出结果如下所示：

```
>>>
===== RESTART: D:\Python35\5.2.py =====
['ypythonh', 'spythono', 'rpythone']
```

#### 4. re.sub() 函数

如果，想根据正则表达式来实现替换某些字符串的功能，我们可以使用 re.sub() 函数实现。

re.sub() 函数的格式如下：

```
re.sub(pattern,rep,string,max)
```

其中，第一个参数为对应的正则表达式，第二个参数为要替换成的字符串，第三个参数为源字符串，第四个参数为可选项，代表最多替换的次数，如果忽略不写，则会将符合模式的结果全部替换。

使用 re.sub() 这个函数，会根据正则表达式 pattern，从源字符串 string 查找出符合模式的结果，并替换为字符串 rep，最多可替换 max 次。

我们可以通过以下实例更好地理解：

```
import re
string="hellomypythonhispythonourpythonend"
pattern="python."
result1=re.sub(pattern,"php",string) # 全部替换
result2=re.sub(pattern,"php",string,2) # 最多替换两次
```

```
print(result1)
print(result2)
```

该实例的输出结果如下：

```
>>>
===== RESTART: D:\Python35\5.2.py =====
hellomyphpisphpurphnd
hellomyphpisphpurpythonend
```

可以看到，在第一行输出结果中，由于没有设置第四个参数，所以会将符合模式的结果全部替换掉，而在第二行输出结果中，由于设置了第四个参数的值为 2，所以，其最多只能替换两次。

下面我们分析一些正则表达式中常用的函数，使用这些函数可以实现各种各样的功能。

## 5.4 常见实例解析

接下来，我们一起来分析 3 个常见的正则表达式的实例。

### 实例 1：匹配 .com 或 .cn 后缀的 URL 网址

实例目的：将一串字符串里面以 .com 或 .cn 为域名后缀的 URL 网址匹配出来，过滤掉其他的无关信息。

实例代码实现：

```
import re
pattern="[a-zA-Z]+://[^\s]*[.com|.cn]"
string="<a href='http://www.baidu.com'> 百度首页 </a>"
result=re.search(pattern,string)
print(result)
```

代码执行结果：

```
>>>
===== RESTART: D:\Python35\5.4.py =====
<_sre.SRE_Match object; span=(9, 29), match='http://www.baidu.com'>
```

实例代码分析：上述代码中，关键点在于正则表达式的设置，要构建好相对完善的正则表达式，则需要观察我们需求的这些信息的规律。首先，“://”是固定的，可以写出来，然后我们要以 .com 或者 .cn 结尾，所以正则表达式的最后应该是 [.com|.cn]，在“://”与 [.com|.cn] 之间，不能出现空格，所以我们可以写为 [^\s]\*，在“://”之前，必须要有内容，所以此时至少要有一次重复，故而我们用“+”而不用“\*”，这些内容可以是任意的字母的组合，包括大小写，所以可以写为 [a-zA-Z]，组合起来，正则表达式就是：“[a-zA-Z]+://[^\s]\*[.com|.cn]”。

然后，我们设置了要检索的源字符串 string，并通过函数进行了检索匹配，成功将对应

的 URL 网址信息提取了出来。

### 实例 2：匹配电话号码

实例目的：将一字符串里面出现的电话号码信息提取出来，过滤掉其他无关信息。

实例代码实现：

```
import re
pattern="\d{4}-\d{7}|\d{3}-\d{8}" # 匹配电话号码的正则表达式
string="021-6728263653682382265236"
result=re.search(pattern,string)
print(result)
```

代码执行结果：

```
>>>
===== RESTART: D:\Python35\5.4.py =====
<_sre.SRE_Match object; span=(0, 12), match='021-67282636'>
```

实例代码分析：本实例中关键点为电话号码正则表达式的设置，我们知道电话号码的区号有的有 3 位数字，有的有 4 位数字，如果区号为 3 位，那么区号后面的数字为 8 位，如果区号为 4 位，那么区号后面的数字为 7 位，区号与后面的数字之间通过“-”连接。所以此时，正则表达式可以设置为“\d{4}-\d{7}|\d{3}-\d{8}”。设置好正则表达式后，可以通过 re.search() 函数将符合条件的信息匹配提取出来。

### 实例 3：匹配电子邮件地址

实例目的：将一字符串里面出现的电子邮件信息提取出来，过滤掉其他无关信息。

实例代码实现：

```
import re
pattern="\w+([+-]\w+)*@\w+([+-]\w+)*\.\w+([+-]\w+)*" # 匹配电子邮件的正则表达式
string="<a href='http://www.baidu.com'>百度首页</a><br><a href='mailto:c-e+o@iqi-  
anyue.com.cn'>电子邮件地址</a>"
result=re.search(pattern,string)
print(result)
```

代码执行结果：

```
>>>
===== RESTART: D:\Python35\5.4.py =====
<_sre.SRE_Match object; span=(59, 81), match='c-e+o@iqi-anyue.com.cn'>
```

实例代码分析：本实例中，关键点还是在于正则表达式的设置，要设置好匹配电子邮件的正则表达式，则需要观察电子邮件的格式，然后总结出一套模式出来。简单提示一下，首先确定电子邮件中的“@”符号，然后观察 @ 后可以出现哪些字符，“@”前可以出现哪些字符，并通过正则表达式描述出来。设置好正则表达式之后，同样可以通过 re.search() 函数将符合条件的信息匹配提取出来。



以上是我们为大家讲解的 3 个正则表达式的应用实例，希望可以抛砖引玉，让大家能够理解并学会对应正则表达式的应用。在构建正则表达式之前，首先需要观察我们需求的信息的规律，根据这个规律用正则表达式描述出来，当然，你也可以在互联网中收集一些前人总结的常见功能的正则表达式，写作案例为自己所用。在实际情况中，没有最好的正则表达式，很多正则表达式在实际应用的时候如果出现一些问题，则需要不断的改进。

## 5.5 什么是 Cookie

在爬虫的使用中，如果涉及登录等操作，经常会使用到 Cookie。

那么什么是 Cookie 呢？

简单来说，我们访问每一个互联网页面，都是通过 HTTP 协议进行的，而 HTTP 协议是一个无状态协议，所谓的无状态协议即无法维持会话之间的状态。

比如，仅使用 HTTP 协议的话，我们登录一个网站的时候，假如登录成功了，但是当我们访问该网站的其他网页的时候，该登录状态则会消失，此时还需要再登录一次，只要页面涉及更新，就需要反复的进行登录，这是非常不方便的。

所以此时，我们需要将对应的会话信息，比如登录成功等信息通过一些方式保存下来，比较常用的方式有两种：通过 Cookie 保存会话信息或通过 Session 保存会话信息。我们分别说一下这两种方式。

如果是通过 Cookie 保存会话信息，此时会将所有的会话信息保存在客户端，当我们访问同一个网站的其他页面的时候，会从 Cookie 中读取对应的会话信息，从而判断目前的会话状态，比如可以判断是否已经登录等。显然，这一种方式我们会用到 Cookie。

如果是通过 Session 保存会话信息，会将对应的会话信息保存在服务器端，但是服务器端会给客户端发 SessionID 等信息，这些信息一般存在客户端的 Cookie 中，当然，如果客户端禁止了 Cookie，也会通过其他方式存储。但是，目前来说，大部分的情况还是会将这一部分的信息存到 Cookie 中。然后，用户在访问该网站其他网页的时候，会从 Cookie 中读取这一部分信息，然后从服务器中的 Session 中根据这一部分 Cookie 信息检索出该客户端的所有会话信息，然后进行会话控制。显然，使用 Session 的方式来保存会话信息，大部分的时候，还是会到 Cookie。

通过前面的分析，可以看到，不管是通过哪种方式进行会话控制，在大部分时候，都会用到 Cookie。比如在爬虫的登录中，如果没有 Cookie，我们登录成功了一个网页，但如果我们要爬去该网站的其他网页的时候，仍然会是未登录状态，如果有了 Cookie，当我们登录成功后，爬取该网站的其他网页时，则会保持登录状态进行内容的爬取。

## 5.6 Cookiejar 实战精析

如果要使用 Python 处理 Cookie，在 Python3 中可以使用 Cookiejar 库进行处理，而在

Python2 则可以使用 CookieLib 库进行处理。

为了了解 Python 中 Cookie 处理的重要性，我们先来看一下，假如没有 Cookie 处理会发生什么事情？

我们可以通过以下实例进行分析。

比如，我们想登录 ChinaUnix 论坛，首先需要找到对应的登录地址，可以看到，登录界面的地址是：<http://bbs.chinaunix.net/member.php?mod=logging&action=login&logsubmit=yes>，如图 5-1 所示。



图 5-1 ChinaUnix 登录界面

但是，该地址却不是真实的登录地址，如果要获得真实的登录地址，我们需要进行分析，分析方法主要有两种，第一种方法是通过 F12 调出调试界面进行分析，第二种方法是使用工具软件进行分析，常用的工具软件有 Fiddler。在此，我们先使用 F12 调试界面的方式进行分析，在第 7 章中，我们会具体讲到 Fiddler 工具的使用。

要获取真实登录地址，我们首先按 F12，调出对应的调试界面，然后，在登录的输入框中输入对应的用户名和密码，如图 5-2 所示。

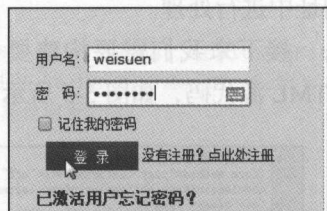


图 5-2 输入对应的用户名和密码

输入了对应账号和密码之后，单击登录，与此同时，观察调试界面，我们可以看得到，调试界面中出现了很多网址，这些网址中，很多都使用 GET 方法，只有一个使用了 POST 方法，这个使用了 POST 方法的网址，在里面就可以分析出处理 POST 表单的真实页面，如图 5-3 所示。我们单击使用 POST 方法的对应网址。

单击进去之后，出现如图 5-4 所示界面，可以看到，有一个 Request URL 的字段，该字段所对应的值就是真实的处理 POST 表单的网址。

我们将其复制出来，即：<http://bbs.chinaunix.net/member.php?mod=logging&action=login>



可以看到, 登录框中用户名所对应的表单 name 为: username, 密码所对应的表单 name 为: password。所以此时, 我们构建的数据中需要包含这两项, 该数据可以构建为:

```
{
  "username": "weisuen",
  "password": "aA123456"
}
```

对应的, “weisuen” 为登录的用户名, “aA123456” 为登录的密码。读者在实际操作该网页时, 换成自己注册的账号名和密码即可。如果要登录其他网页, 按以上的流程进行分析即可得到对应的真实的登录网址以及构建数据时所需要的个字段名。

有了这些信息之后, 就可以进行登录了, 首先进行无 Cookie 处理的登录, 代码如下所示:

```
import urllib.request
import urllib.parse
url = "http://bbs.chinaunix.net/member.php?mod=logging&action=login&loginsubmit=yes&loginhash=L768q"
postdata =urllib.parse.urlencode({
  "username": "weisuen",
  "password": "aA123456"
}).encode('utf-8') # 使用 urlencode 编码处理后, 再设置为 utf-8 编码
req = urllib.request.Request(url,postdata)# 构建 Request 对象
req.add_header('User-Agent', 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0')
data=urllib.request.urlopen(req).read() # 登录并爬取对应网页
fhandle=open("D:/Python35/myweb/part5/1.html", "wb")
fhandle.write(data)# 将内容写入对应文件
fhandle.close()

url2="http://bbs.chinaunix.net/" # 设置要爬取的该网站下其他网页地址
req2 = urllib.request.Request(url2,postdata)
req2.add_header('User-Agent', 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0')
data2=urllib.request.urlopen(req2).read()# 爬取该站下的其他网页
fhandle=open("D:/Python35/myweb/part5/2.html", "wb")
fhandle.write(data2)# 将爬取到的其他网页写入对应文件
fhandle.close()
```

以上代码中, 设置好登录网址 (注意此时需要真实的处理 POST 表单的登录网址) 和构建好数据之后, 使用 urllib.request.Request() 创建了一个 Request 对象, 然后添加了对应的头信息, 随后使用 urllib.request.urlopen() 进行登录并爬取了对应的网页信息保存到本地的文件 1.html 中, 然后再爬取了该站下的其他网页并保存到本地文件 2.html 中。

执行后, 我们去目录 "D:/Python35/myweb/part5/" 下察看爬取到的网页, 如图 5-6 所示。

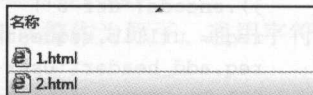


图 5-6 爬取到的网页



首先打开爬取到的第一个网页，如图 5-7 所示。

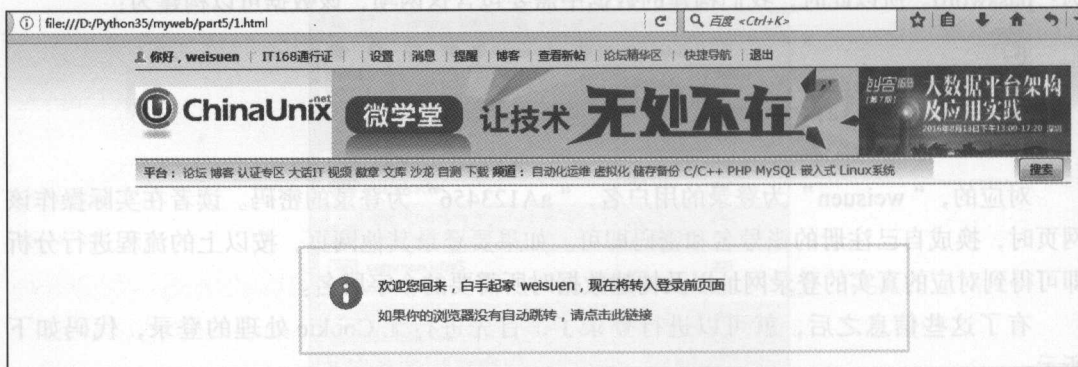


图 5-7 打开爬取到的第一个网页

此时已经成功登录了。然后再打开爬取到的第二个网页，如图 5-8 所示：

可以发现，在爬取到的第二个网页中，仍然需要我们进行登录，此时是未登录状态。

为什么会出现这种情况呢？是因为在这里，我们没有设置 Cookie 处理，而 HTTP 协议是一个无状态协议，我们访问了新网页，自然会话信息会消失。

如果希望登录状态一直保持，则需要进行 Cookie 处理。

进行 Cookie 处理的一种常用思路如下：

- 1) 导入 Cookie 处理模块 `http.cookiejar`。
- 2) 使用 `http.cookiejar.CookieJar()` 创建 `CookieJar` 对象。
- 3) 使用 `HTTPCookieProcessor` 创建 cookie 处理器，并以其为参数构建 `opener` 对象。
- 4) 创建全局默认的 `opener` 对象。

所以，上面一个实例的代码可以改进为：

```
import urllib.request
import urllib.parse
import http.cookiejar
url = "http://bbs.chinaunix.net/member.php?mod=logging&action=login&loginsubmit=yes&loginhash=L768q"
postdata = urllib.parse.urlencode({
    "username": "weisuen",
    "password": "aA123456"
}).encode('utf-8')
req = urllib.request.Request(url, postdata)
req.add_header('User-Agent', 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0')
# 使用 http.cookiejar.CookieJar() 创建 CookieJar 对象
```

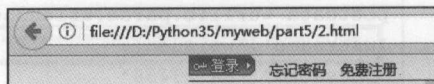


图 5-8 打开爬取到的第二个网页

```

cjar=http.cookiejar.CookieJar()
# 使用 HTTPCookieProcessor 创建 cookie 处理器, 并以其为参数构建 opener 对象
opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cjar))
# 将 opener 安装为全局
urllib.request.install_opener(opener)
file=opener.open(req)
data=file.read()
file=open("D:/Python35/myweb/part5/3.html", "wb")
file.write(data)
file.close()
url2="http://bbs.chinaunix.net/"
data2=urllib.request.urlopen(url2).read()
fhandle=open("D:/Python35/myweb/part5/4.html", "wb")
fhandle.write(data2)
fhandle.close()

```

可以看到, 上述代码通过 `cjar=http.cookiejar.CookieJar()` 创建了一个 `CookieJar` 对象 `cjar`, 然后再通过 `urllib.request.HTTPCookieProcessor(cjar)` 创建了一个 `Cookie` 处理器, 随后以该 `Cookie` 处理器为参数通过 `urllib.request.build_opener()` 创建了一个 `opener` 对象, 并使用 `urllib.request.install_opener(opener)` 创建全局默认的 `opener` 对象, 那么这样, 在使用 `urlopen()` 时, 亦会使用我们安装的 `opener` 对象。然后, 我们进行登录并爬取了对应网页保存到本地文件 3.html 中, 接着爬取了该站下的其他网页并保存到本地文件 4.html 中。

我们用浏览器打开本地文件 3.html, 发现已经登录成功, 如图 5-9 所示:

接着, 用浏览器打开本地文件 4.html, 该文件为爬虫爬取的网站下的其他页面, 如图 5-10 所示。可以看到, 此时为登录状态, 也就是说, 对应的登录状态已经通过 `Cookie` 保存, 此时, 成功完成了 `Cookie` 处理, 并保持了爬虫在对应站点的登录状态信息。

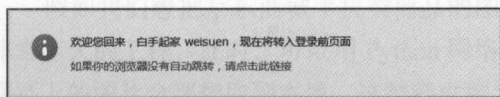


图 5-9 登录成功

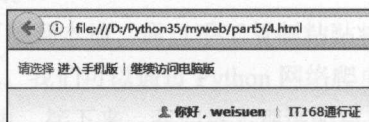


图 5-10 保持爬虫在对应站点的登录状态

## 5.7 小结

1) 有时我们在进行字符串处理的时候, 希望按自定义的规则进行处理, 我们将这些规则称为模式。可以用正则表达式来描述这些自定义规则, 正则表达式也称为模式表达式。

2) 在 Python 中, 一般我们会使用 `re` 模块实现 Python 正则表达式的功能。

3) 正则表达式中常见的原子有: 普通字符作为原子、非打印字符作为原子、通用字符作为原子、原子表。

4) 模式修正符, 可以在不改变正则表达式的情况下, 通过模式修正符改变正则表达式

的含义，从而实现一些匹配结果的调整等功能。

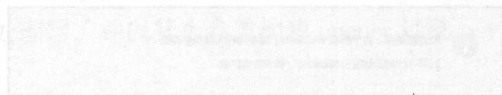
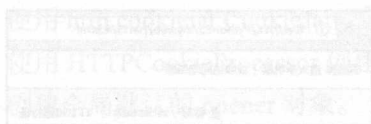
5) 我们访问每一个互联网页面，都是通过 HTTP 协议进行的，而 HTTP 协议是一个无状态协议，所谓的无状态协议即无法维持会话之间的状态。

6) 会话信息控制比较常用的方式有两种：通过 Cookie 保存会话信息、通过 Session 保存会话信息。

7) 如果是通过 Session 保存会话信息，会将对应的会话信息保存在服务器端，但是服务器端会给客户端发 SessionID 等信息，这些信息一般存在客户端的 Cookie 中，当然，如果客户端禁止了 Cookie，也会通过其他方式存储。但是，目前来说，大部分的情况还是会将这一部分的信息存到 Cookie 中。然后，用户在访问该网站其他网页的时候，会从 Cookie 中读取这一部分信息，然后从服务器中的 Session 中根据这一部分 Cookie 信息检索出该客户端的所有会话信息，然后进行会话控制。显然，使用 Session 的方式来保存会话信息，大部分的时候，还是会用的 Cookie。

8) 如果要使用 Python 处理 Cookie，在 Python3 中可以使用 cookiejar 库进行处理，而在 Python2 则可以使用 cookielib 库进行处理。

9) 如果要获得真实的登录地址，我们需要进行分析，分析方法主要有两种，第一种方法是通过 F12 调出调试界面进行分析，第二种方法是使用工具软件进行分析，常用的工具软件有 Fiddler。



## 手写 Python 爬虫

通过前面几章的学习,相信大家对网络爬虫的基础知识已经有所掌握,在本章中,我们将由浅入深,带领大家从零开始手写一些常用的 Python 网络爬虫,包括图片爬虫、链接爬虫、糗事百科爬虫、微信爬虫、多线程爬虫等。

### 6.1 图片爬虫实战

首先,我们讲解一个比较简单的 Python 网络爬虫。

假如我们想把京东商城手机类商品的图片全部下载到本地,通过手工复制粘贴将是一项非常庞大的工程,此时,可以用 Python 网络爬虫实现。我们可以通过 Python 网络爬虫将这些网页上的图片全部爬取到本地,这类爬虫称为图片爬虫,接下来,我们将实现该爬虫。

首先打开要爬取的第一个网页,这个网页将作为要爬取的起始页面。比如,我们要爬取京东商城手机类商品的图片,就需要打开京东商城首页(www.jd.com),然后选择对应的“手机、数码、京东通信”分类,并进入其下的“手机”子分类,如图 6-1 所示。

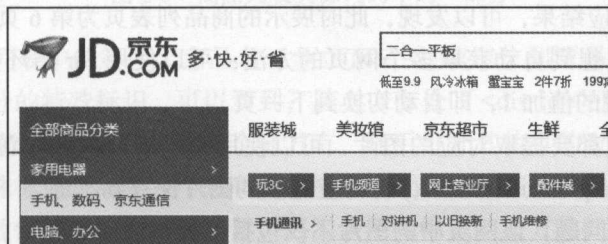


图 6-1 京东首页的手机子分类



进入之后,对应的网址即为我们要爬取的第一个网页,对应的界面如图 6-2 所示:

我们将对应的网址提取出来,为: <http://list.jd.com/list.html?cat=9987,653,655>。接下来,对网址进行分析,此时我们只获得了商品列表的第一个页面,往下拖动,可以看到,该类目下商品有很多页,如图 6-3 所示。

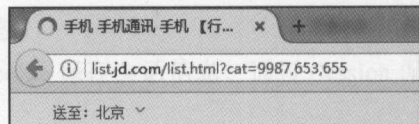


图 6-2 手机分类网页的网址



图 6-3 手机分类下的商品列表

那么,怎样才能自动爬取第一页以外的其他页面呢?

可以单击“下一页”,观察网址的变化。在单击了下一页之后,发现网址变成了 [http://list.jd.com/list.html?cat=9987,653,655&page=2&trans=1&JL=6\\_0\\_0&ms=5#J\\_main](http://list.jd.com/list.html?cat=9987,653,655&page=2&trans=1&JL=6_0_0&ms=5#J_main)。可以发现,在这里要获取第几页是通过 URL 网址识别的,即通过 GET 的方式请求的。在这个 GET 请求中,有多个字段,其中有一个字段为 `page`,对应值为 2,由此,我们可以得到该网址中的关键信息为: <http://list.jd.com/list.html?cat=9987,653,655&page=2>,而根据规律可以推测 `page` 字段代表的是获取第几页的商品列表。接下来,我们可以根据推测进行相应的验证,可以将 `page=2` 换成 `page=6`,即网址变为: <http://list.jd.com/list.html?cat=9987,653,655&page=6>,在浏览器上输入该网址并观察对应结果,可以发现,此时展示的商品列表页为第 6 页。

由此,我们可以想到自动获取多个网页的方法:可以使用 `for` 循环实现,每次循环后,对应网址中 `page` 字段的值加 1,即自动切换到下一页。

在每页中,我们都要提取对应的图片,可以使用正则表达式匹配源码中图片的链接部分,然后通过 `urllib.request.urlretrieve()` 将对应链接的图片保存到本地。

但是这里有一个问题,该网页中的图片不仅包括列表中商品的图片,还包括旁边的一些无关图片,如图 6-4 所示:

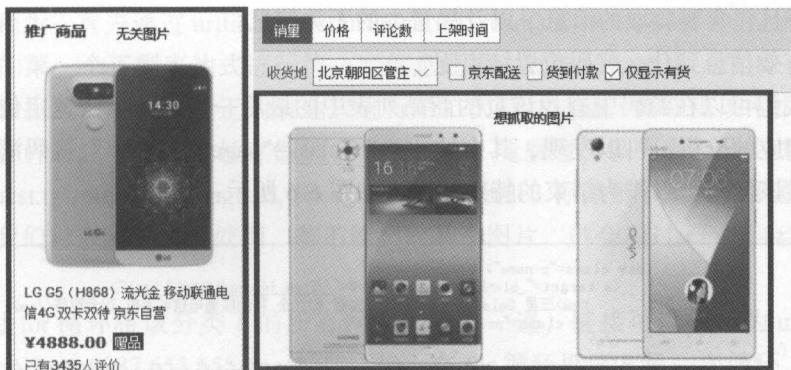


图 6-4 商品列表页中的无关信息与要爬取信息示意图

所以我们可以先进行一次信息过滤，第一次信息过滤将中间的商品列表部分数据留下，将其他部分的数据过滤掉。可以单击右键，然后查看网页的源代码，如图 6-5 所示：

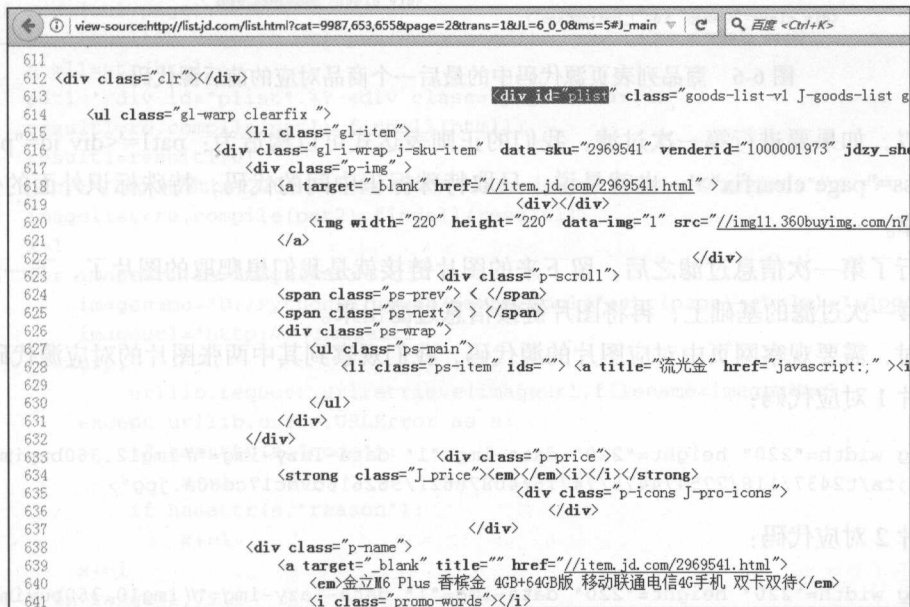


图 6-5 商品列表页源代码中的一部分

可以通过商品列表中的第一个商品名“金立 M6”快速定位到源码中的对应位置，然后观察其商品列表部分的特殊标识，可以看到，其上方有处“<div id="plist">”的代码，然后我们在源码中搜索该代码，发现只有一个地方有，随后打开其他页的对应页面，发现仍然具有此规律，即该特殊标识距离第一个想爬取的图片较近，并且在页面中是唯一的，说明该特殊标识可以作为有效信息的起始过滤位置，因为该特殊标识的上方的代码都不是我们关注的内容，自然可以过滤掉。当然，你也可以用其他的代码作为特殊标识，但是该特殊标识要满

足唯一性，并且包含要爬取的信息，以及尽量少的无关信息。

那么，有效信息到什么代码位置结束呢？

同样，我们可以在源码中查找该页的商品列表中的最后一个商品名快速定位到对应的代码，并进行相应的分析，可以发现，其中的“<div class="page clearfix"”代码满足特殊标识的要求，所以该代码可以作为结束的特殊标识，如图 6-6 所示：

```

2761                                     </div>
2762     <div class="p-name">
2763         <a target="_blank" title="" href="//item.id.com/1411053.html">
2764             <em>三星 Galaxy S6 (G9200) 32G版 铂光金 移动联通电信4G手机 双卡双待</em>
2765             <i class="promo-words"></i>
2766         </a>
2767     </div>
2768         <div class="p-commit"></div>
2769         <div class="p-focus"><a class="J_focus" data-sku="1411053" href=
2770         <div class="p-shop" data-score="4" data-reputation="0" data-shopid="" data-shop_nam
2771         <div class="p-stock" data-isDeliverable="5" style="display: n
2772     </div>
2773 </li>
2774
2775 </ul>
2776 <div class="clr"></div>
2777 </div>
2778                                     <div class="page clearfix">
2779     <div class="p-wrap" id="J_bottomPage">
2780         <span class="p-num">

```

图 6-6 商品列表页源代码中的最后一个商品对应的前后源代码

所以，如果要进行第一次过滤，我们的正则表达式可以构造为：pat1='<div id="plist".+?<div class="page clearfix">'。也就是说，只取特殊标识中间的代码，特殊标识外面的代码全部过滤掉。

进行了第一次信息过滤之后，留下来的图片链接就是我们想爬取的图片了，下一步我们需要在第一次过滤的基础上，再将图片链接信息过滤出来。

此时，需要观察网页中对应图片的源代码，我们观察到其中两张图片的对应源代码。

图片 1 对应代码：

```
<img width="220" height="220" data-img="1" data-lazy-img="//img12.360buyimg.com/
n7/jfs/t2437/118/775474476/74776/4087862f/562616d9Nc17cd80a.jpg">
```

图片 2 对应代码：

```
<img width="220" height="220" data-img="1" data-lazy-img="//img10.360buyimg.com/
n7/jfs/t2230/83/2893465811/162158/80a547ef/56fa0f30N7794db4a.jpg">
```

对比这两张图片的代码，并观察其中的规律，发现其基本格式是一样的，只是图片链接网址不一样，所以此时，我们可以根据该规律构造出提取图片链接的正则表达式：

```
pat2='<img width="220" height="220" data-img="1" data-lazy-img="//(.+?.jpg)">'
```

我们根据该正则表达式，则可以提取出一个页面中所有想要爬取的图片的链接。

所以，根据上面的分析，我们可以得到该爬虫的编写思路与过程，具体如下：

1) 建立一个爬取图片的自定义函数，该函数负责爬取一个页面下的我们想爬取的图

片，爬取过程为：首先通过 `urllib.request.urlopen(url).read()` 读取对应网页的全部源代码，然后根据上面的第一个正则表达式进行第一次信息过滤，过滤完成之后，在第一次过滤结果的基础上，根据上面的第二个正则表达式进行第二次信息过滤，提取出该网页上所有目标图片的链接，并将这些链接地址存储的一个列表中，随后遍历该列表，分别将对应链接通过 `urllib.request.urlretrieve (imageurl, filename=imageurl)` 存储到本地，为了避免程序中途异常崩溃，我们可以建立异常处理，若不能爬取某个图片，则会通过 `x+=1` 自动跳到下一个图片。

2) 通过 `for` 循环将该分类下的所有网页都爬取一遍，链接可以构造为 `url="http://list.jd.com/list.html?cat=9987,653,655&page="+str(i)`，在 `for` 循环里面，每一次循环，对应的 `i` 会自动加 1，每次循环的时候通过调用 1) 中的函数实现该页图片的爬取。

完整的代码如下所示：

```
import re
import urllib.request
def crawl(url, page):
    html1=urllib.request.urlopen(url).read()
    html1=str(html1)
    pat1='<div id="plist".*? <div class="page clearfix">'
    result1=re.compile(pat1).findall(html1)
    result1=result1[0]
    pat2='<img width="220" height="220" data-img="1" data-lazy-img="//(.+?.jpg)">'
    imagelist=re.compile(pat2).findall(result1)
    x=1
    for imageurl in imagelist:
        imagename="D:/Python35/myweb/part6/img1/"+str(page)+str(x)+".jpg"
        imageurl="http://"+imageurl
        try:
            urllib.request.urlretrieve(imageurl,filename=imagename)
        except urllib.error.URLError as e:
            if hasattr(e,"code"):
                x+=1
            if hasattr(e,"reason"):
                x+=1
            x+=1
    for i in range(1,79):
        url="http://list.jd.com/list.html?cat=9987,653,655&page="+str(i)
        crawl(url,i)
```

代码中，我们会爬取第 1 页到第 78 页的所有目标图片，并且将图片存储到本地目录 "D:/Python35/myweb/part6/img1/" 下，图片在本地的名字为“页号+顺序号.jpg”。

运行该程序，在几分钟后，结果如图 6-7 所示。

可以发现，几分钟的工夫，爬虫就已经自动在对应网址中爬取了 3213 张手机图片。

通过这个爬虫项目的学习，相信大家已经初步学会如何手写一个图片网络爬虫项目了。



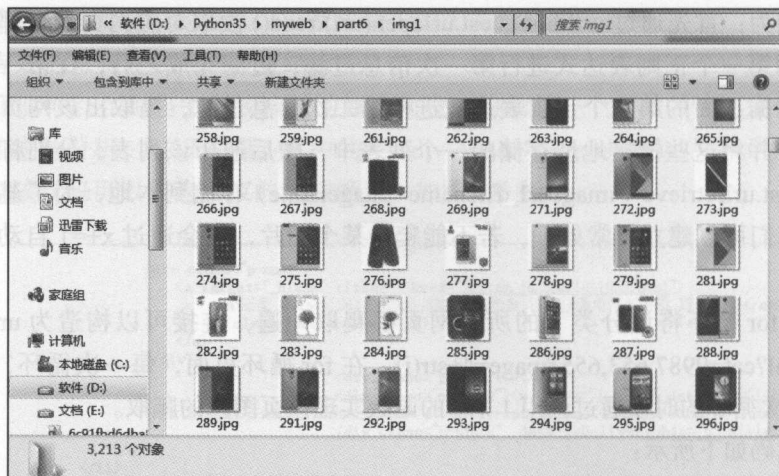


图 6-7 爬取到本地的京东手机分类下商品的图片

## 6.2 链接爬虫实战

假设我们想把一个网页中所有的链接地址提取出来，那么此时我们可以使用 Python 链接爬虫来实现。

本项目链接爬虫实现的思路如下：

- 1) 确定好要爬取的入口链接。
- 2) 根据需求构建好链接提取的正则表达式。
- 3) 模拟成浏览器并爬取对应网页。
- 4) 根据 2) 中的正则表达式提取出该网页中包含的链接。
- 5) 过滤掉重复的链接。
- 6) 后续操作。比如打印这些链接到屏幕上等。

比如，要获取 "http://blog.csdn.net/" 网页上的所有链接，我们可以通过如下程序实现：

```
# 爬取页面链接
import re
import urllib.request
def getlink(url):
    # 模拟成浏览器
    headers=("User-Agent","Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0")
    opener = urllib.request.build_opener()
    opener.addheaders = [headers]
    # 将 opener 安装为全局
    urllib.request.install_opener(opener)
    file=urllib.request.urlopen(url)
    data=str(file.read())
```

```

# 根据需求构建好链接表达式
pat='(https?:/{0,2}(\w|\/)*)'
link=re.compile(pat).findall(data)
# 去除重复元素
link=list(set(link))
return link
# 要爬取的网页链接
url="http://blog.csdn.net/"
# 获取对应网页中包含的链接地址
linklist=getlink(url)
# 通过 for 循环分别遍历输出获取到的链接地址到屏幕上
for link in linklist:
    print(link[0])

```

在程序中，我们建立了自定义函数 `getlink(url)`，专门负责爬取对应 URL 网页上的所有链接。在函数中，首先需要设置 `header` 信息等模拟成浏览器，然后需要读取该网页的源代码，读取了源代码之后根据构建的正则表达式，通过 `re.compile(pat).findall(data)` 提取出该页面中的所有链接，此时提取到的链接可能会重复，所以我们可以通过 `list(set(link))` 将对应重复的元素过滤掉，然后返回过滤后的列表。在函数外，首先需要定义好要爬取的网页链接，随后可以将获取到的链接通过 `for` 循环遍历出来并打印的屏幕上。

程序的执行结果如下：

```

>>>
===== RESTART: D:\Python35\6.2.py =====
http://blog.csdn.net/han_xiaoyang/article/details/52275318
http://blog.csdn.net/linshuhei/article/details/52062969
http://blog.csdn.net/experts/rule.html
http://blog.csdn.net/wzy_1988/article/details/52277315
http://img.knowledge.csdn.net/upload/base/1471395508721_721.jpg
http://csdnimg.cn/public/common/libs/bootstrap/css/bootstrap.css
http://c.csdnimg.cn/public/common/toolbar/js/toolbar.js
http://blog.csdn.net/u013132051/article/details/52250507
http://lib.csdn.net/base/go
http://img.blog.csdn.net/20160711171908566
http://static.blog.csdn.net/static/css/font-awesome.min.css
http://static.blog.csdn.net/static/images/back_top.png
http://avatar.csdn.net/2/3/7/2_dongdong9223.jpg
http://blog.csdn.net/beliefer/article/details/50499169
http://blog.csdn.net/u012702547
.....

```

由于输出的链接过多，鉴于篇幅关系，在此我们省略掉了一部分输出结果。可以看到，我们通过该程序获取到了网页 "`http://blog.csdn.net/`" 中所包含的所有链接。其中比较关键的一点在于，正则表达式的构造不一定是固定的，需要根据自身的需求来进行调整，并且可以不断完善，下面简要说一下上述程序中链接提取的正则表达式的含义。上面的正则表达式为 `'(https?:/{0,2}(\w|\/)*)'`，确定其简单版链接的基本格式为：“`http://xxx.yyy`”，其中

“xxx”和“yyy”均代表可变化部分。根据该简单版本完善,首先,协议部分有的是“http”,有的是“https”,此时“s”可有可无,然后“xxx”部分不可以出现空格,不可以出现双引号,也不可以出现分号,“yyy”部分是一些非特殊字符,也可以是“/”符号。当然,该正则表达式仍有完善的空间,在正则表达式的思维中,只有更好,没有最好。

## 6.3 糗事百科爬虫实战

假如我们想爬取糗事百科(<http://www.qiushibaike.com/>)上的段子,也可以编写对应的Python网络爬虫实现。

本项目糗事百科网络爬虫的实现思路及步骤如下:

1) 分析各页间的网址规律,构造网址变量,并可以通过for循环实现多页内容的爬取。

2) 构建一个自定义函数,专门用来实现爬取某个网页上的段子,包括两部分内容,一部分是对应用户,一部分是用户发表的段子内容。该函数功能实现的过程为:首先,模拟成浏览器访问,观察对应网页源代码中的内容,将用户信息部分与段子内容部分的格式写成正则表达式。随后,根据各正则表达式分别提取出该页中所有的用户与所有的内容,然后通过for循环遍历段子内容并将内容分别赋给对应的变量,这里变量名是有规律的,格式为“content+ 顺序号”,接下来再通过for循环遍历对应用户,并输出该用户对应的内容。

3) 通过for循环分别获取多页的各页URL链接,每页分别调用一次getcontent(url, page)函数。

具体的实现代码如下:

```
import urllib.request
import re
def getcontent(url,page):
    # 模拟成浏览器
    headers=("User-Agent","Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0")
    opener = urllib.request.build_opener()
    opener.addheaders = [headers]
    # 将 opener 安装为全局
    urllib.request.install_opener(opener)
    data=urllib.request.urlopen(url).read().decode("utf-8")
    # 构建对应用户提取的正则表达式
    userpat='target="_blank" title="(.*?)>'
    # 构建段子内容提取的正则表达式
    contentpat='<div class="content">(.*?)</div>'
    # 寻找出所有的用户
    userlist=re.compile(userpat,re.S).findall(data)
    # 寻找出所有的内容
    contentlist=re.compile(contentpat,re.S).findall(data)
    x=1
    # 通过 for 循环遍历段子内容并将内容分别赋给对应的变量
```

```

for content in contentlist:
    content=content.replace("\n","")
    # 用字符串作为变量名, 先将对应字符串赋给一个变量
    name="content"+str(x)
    # 通过 exec() 函数实现用字符串作为变量名并赋值
    exec(name+'=content')
    x+=1

y=1
# 通过 for 循环遍历用户, 并输出该用户对应的内容
for user in userlist:
    name="content"+str(y)
    print(" 用户 "+str(page)+str(y)+" 是:"+user)
    print(" 内容是:")
    exec("print('"+name+"')")
    print("\n")
    y+=1

# 分别获取各页的段子, 通过 for 循环可以获取多页
for i in range(1,30):
    url="http://www.qiushibaike.com/8hr/page/"+str(i)
    getcontent(url,i)

```

执行结果如下:

```

>>>
===== RESTART: D:\Python35\6.3.py =====
用户 12 是: 哈哈啊哈哈啊哈哈啊哈哈啊哈哈
内容是:
LZ 农村的, 刚刚在地里干活, 下暴风雨, 雨有多大呢? 我跑进一个破房子里躲雨, 不一会儿一只松鼠也进来了, 看到我, 愣了一下。看了一眼外面的雨, 又看看我, 还是蹲下来跟我一起避雨……
用户 13 是: 白子国王
内容是:
你知道神经大条有多粗吗! 女同学一枚, 第一次带男朋友回老家, 有点忐忑, 于是叫男友在村口等着, 她先回家与家人交接好再出来接他。不料家里亲朋太热闹, 竟然忘了带男友回家这茬事。结果她男友一直在村口从中午等到天黑, 靠着草垛睡着了。
.....

```

由于爬取到的内容较多, 篇幅有限, 故而输出结果有所省略, 我们可以手动打开对应网页, 看到此时获取到的内容是一致的, 如图 6-8 所示, 所以此时爬取成功。

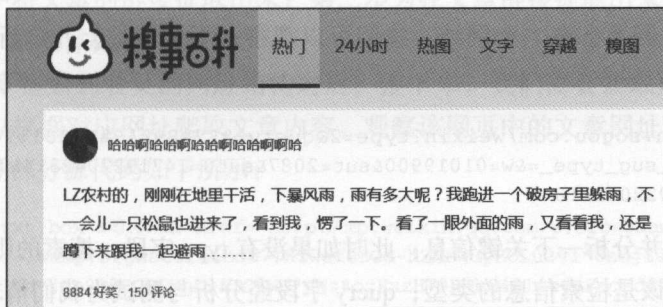


图 6-8 爬取的内容与网站源内容对比



通过爬虫进行自动化爬取，可以为我们省下很多事情。比如，有些站长需要采集一些内容到自己的网站上，如果通过复制粘贴的方式，耗费的精力非常大，而采用爬虫的方式，我们可以直接将关注的数据库爬取下来，并可以用程序直接自动写进对应的数据库中，此时，网站上的内容就可以实现自动更新了。当然，学习爬虫需要遵守法律，比如一些别人原创性的内容，我们应当声明转载，尊重别人的原创成果，对于一些禁止爬取或禁止转载的内容，我们亦应当遵守相关规定。

如果糗事百科网页的代码结构发生变化，那么上述爬虫代码中的 URL 网址以及正则表达式也需要进行相应的改变，否则将无法爬取。所以，我们学习爬虫，一定要学会写爬虫的这一套方法并灵活运用，这样，在具体问题变化了之后，我们能随之做出改变。通过上面 3 个项目例子的学习，相信大家已经发现，写这些爬虫基本思路是一致的，只不过具体细节可能有所不同。希望大家可以参照代码，将上面 3 个例子认真分析并独自写两遍，这样可以更好的掌握 Python 网络爬虫的编写。

## 6.4 微信爬虫实战

接下来，我们为大家讲解如何编写微信爬虫。相对上面 3 个例子来说，微信爬虫会稍难一些，因为在我们爬取微信文章的时候，经常会被官方封杀 IP，被封杀 IP 之后，爬虫就无法获取对应的内容了。

在本项目中，我会给出对应的解决方案并通过实战案例为大家讲解如何解决这个棘手的问题。

假如现在我们希望搜索与某个关键词相关的微信公众平台文章，并且将这些文章的标题与内容全部爬取到本地，应该怎么实现呢？

首先，来看一下我们是如何通过人工方法实现网页爬取的。先了解人工实现的过程，再编写爬虫实现自动化。

可以以搜狗的微信搜索平台 (<http://weixin.sogou.com/>) 为入口，假如我们想了解物联网相关的文章，可以输入对应的关键词并进行搜索，如图 6-9 所示，可以检索到对应的文章列表。

此时，在浏览器上会有一个网址，我们可以观察一下该网址，并分析一下该网址的结构，此时的网址为：

```
http://weixin.sogou.com/weixin?type=2&query=%E7%89%A9%E8%81%94%E7%BD%91&ie=utf8&_sug_&y&_sug_type_&w=01019900&sut=2087&sst0=1471922042314&lkt=1%2C1471922042210%2C1471922042210
```

我们可以测试并分析一下关键信息：此时如果没有 type 字段，搜索的则不是文章，所以 type 字段控制的应该是检索信息的类型；query 字段经分析与测试为我们请求的关键词信息，只不过此时进行了编码；其他的为非关键信息，省略亦不会影响。但此时我们获取的文章列

表仅为第一页，我们可以在网页下方单击下页并观察网址变化。

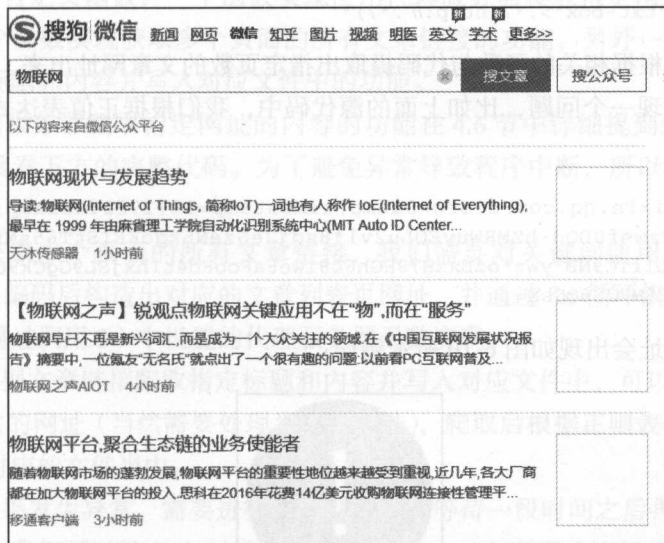


图 6-9 搜狗微信中检索对应关键词后的文章列表页

单击了下一页后，网址变为：

```
http://weixin.sogou.com/weixin?query=%E7%89%A9%E8%81%94%E7%BD%91&_sug_type_=&su
t=2087&lkt=1%2C1471922042210%2C1471922042210&_sug_=y&type=2&sst0=1471922042314&
page=2&ie=utf8&w=01019900&dr=1
```

可以观察到，新网址中比原网址多了 `page` 字段，所以此时控制页数的字段应为 `page`，经过测试，满足分析结果。所以经过上述分析，我们知道，此时的关键字段为 `type`、`query`、`page`，分别控制搜索的类型、关键词以及检索结果中的页码。所以，我们的网址结构可以构造为：`http://weixin.sogou.com/weixin?type=2&query=关键词 &page=页码`。

在了解网址的构造之后，我们可以进行下一步分析。我们现在需要爬取文章的具体内容，所以思路大体可以分为两步来做：第一步可以检索对应关键词得到相应的文章检索结果，并在该页面中将文章的链接提取出来；第二步为在文章链接提取出来之后，根据这些链接地址采集文章中的具体标题和内容。

在分析了文章检索列表页的网址结构之后，接下来，我们需要提取列表页中的文章网址，以便后面可以根据对应网址爬取文章内容。观察该网页中的文章网址部分的源代码，其中一篇文章的网址部分源代码如下所示：

```
<div class="txt-box"><h4><a href="http://mp.weixin.qq.com/s?src=3&timestamp=14719
22485&ver=1&signature=bdb7TKzwef0D0d-hZMRWuyEOLu2v1*8sdj2e5zeN6kgoKFISTt65s
Mt7-hE3W958Wcfh-jnp9Q9ZCRNNnL1fTC9NB*yWz*ozExzH79GHg6481we6aF0oFd4tIhxjSL9GgCk9c00FU
DHT1MHKmc9k87evq9uFPOSEE8M7286QGsM=" target="_blank" id="sogou_vr_11002601_title_0"
uigs_exp_id=""><em><!--red_beg--> 物联网 <!--red_end--></em> 与智能矿山 </a></h4>
```

所以此时，我们可以将提取文章网址的正则表达式构造为：

```
'<div class="txt-box">.*?(http://.*?)'
```

接下来便可以根据相关的函数与代码提取出指定页数的文章网址出来。

但是这里会出现一个问题，比如上面的源代码中，我们根据正值表达式提取出来的对应网址为：

```
http:// mp.weixin.qq.com/s?src=3&timestamp=1471922485&ver=1&signature=bdb7TKzwef0D0d-hZMRWuyEOLu2v1*8sdj2e5zeN6kgoKFIST65sMt7-hE3W958WCfh-jnp9Q9ZCRNNnL1fTC9NB*yWz*ozExzH79HGh6481we6aFooFd4tIhxjSL9GgCk9cO0FUDHT1MHKmc9k87evq9uFPOSEE8M7286QGSM=
```

但是访问该网址会出现如图 6-10 所示错误：

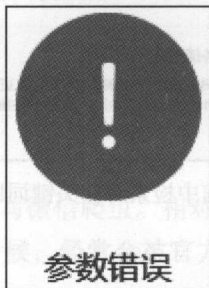


图 6-10 出现参数错误的问题

我们直接通过列表页打开该文章地址，可以打开，发现真实的 url 地址如下：

```
http:// mp.weixin.qq.com/s?src=3&timestamp=1471922485&ver=1&signature=bdb7TKzwef0D0d-hZMRWuyEOLu2v1*8sdj2e5zeN6kgoKFIST65sMt7-hE3W958WCfh-jnp9Q9ZCRNNnL1fTC9NB*yWz*ozExzH79HGh6481we6aFooFd4tIhxjSL9GgCk9cO0FUDHT1MHKmc9k87evq9uFPOSEE8M7286QGSM=
```

网址中的 timestamp 字段代表时间戳，分析时可以忽略，我们对比一下两个网址的区别，就可以发现，提取出来的网址比真实的网址多了一些“&amp;”字符串，故而此时，我们需要对采集出来的网址进行相应处理，处理为真实网址。我们通过 url.replace (“&amp;”, “”) 去掉对应多余的字符串即可。

有了文章网址之后，便可以根据对应的文章网址爬取相应的网页。同样，通过观察文章页与文章页源代码之间的对应关系，可以构建出文章标题和内容的对应正则表达式。

此时，还需要解决一个问题，就是在爬取微信文章的时候，经常会被官方屏蔽 IP，这个问题我们可以采用 4.6 节提到的代理服务器的方式解决，同样，我们可以在 <http://yum.iqianyue.com/proxy> 中获取到最新的代理服务器及端口，并尝试用这些代理服务器爬取对应网页，当然有些新代理服务器也可能失效，在这种情况下可以多试几个，或者通过互联网寻找一些稳定的更新迅速的代理服务器地址。

通过上面的分析，我们可以对该项目进行简单的规划，这样后续实现起来会更有条理，

本项目的规划如下：

1) 建立3个自定义函数：一个函数实现使用代理服务器爬取指定网址并返回爬取到的数据的功能，一个函数实现获取多个页面的所有文章链接的功能，另外一个函数实现根据文章链接爬取指定标题和内容并写入对应文件中的功能。

2) 使用代理服务器爬取指定网址的内容的功能在4.6节中详细提到过，在此不过多分析，各位可以直接看下方的完整代码。为了避免异常导致程序中断，所以在此有必要建立异常处理机制。

3) 要实现获取多个页面的所有文章链接，我们需要对关键词使用 `urllib.request.quote` (key) 进行编码，编码后构造出对应的文章列表页网址，并通过 `for` 循环依次爬取各页的文章链接，爬取时，通过调用2)中设置的代理服务器函数实现。

4) 要实现根据文章链接爬取指定标题和内容并写入对应文件中，可以使用 `for` 循环依次爬取3)中所提供的网址（当然需要处理为真实网址），爬取后根据正则表达式提取出我们关注的内容并写入对应的文件当中。

5) 代码中如果发生异常，需要进行延时处理，即等待一段时间之后再尝试下一次操作。要实现延时处理，我们可以导入 `time` 模块，使用 `time.sleep()` 实现，比如 `time.sleep(7)` 即为延时7秒。

根据上面的规划，我们便可以去具体写对应的程序了，本项目完整的程序代码如下，供大家参考，程序中的重难点代码处笔者都提供了注释。

```
import re
import urllib.request
import time
import urllib.error
# 模拟成浏览器
headers=("User-Agent","Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0")
opener = urllib.request.build_opener()
opener.addheaders = [headers]
# 将 opener 安装为全局
urllib.request.install_opener(opener)
# 设置一个列表 listurl 存储文章网址列表
listurl=[]
# 自定义函数，功能为使用代理服务器
def use_proxy(proxy_addr,url):
    # 建立异常处理机制
    try:
        import urllib.request
        proxy= urllib.request.ProxyHandler({'http':proxy_addr})
        opener = urllib.request.build_opener(proxy, urllib.request.HTTPHandler)
        urllib.request.install_opener(opener)
        data = urllib.request.urlopen(url).read().decode('utf-8')
        return data
    except urllib.error.URLError as e:
```



```

if hasattr(e, "code"):
    print(e.code)
if hasattr(e, "reason"):
    print(e.reason)
# 若为 URLError 异常, 延时 10 秒执行
time.sleep(10)
except Exception as e:
    print("exception:" + str(e))
    # 若为 Exception 异常, 延时 1 秒执行
    time.sleep(1)

# 获取所有文章链接
def getlisturl(key, pagestart, pageend, proxy):
    try:
        page = pagestart
        # 编码关键词 key
        keycode = urllib.request.quote(key)
        # 编码 "&page"
        pagecode = urllib.request.quote("&page")
        # 循环爬取各页的文章链接
        for page in range(pagestart, pageend + 1):
            # 分别构建各页的 url 链接, 每次循环构建一次
            url = "http://weixin.sogou.com/weixin?type=2&query=" + keycode + pagecode + str(page)
            # 用代理服务器爬取, 解决 IP 被封杀问题
            data1 = use_proxy(proxy, url)
            # 获取文章链接的正则表达式
            listurlpat = '<div class="txt-box">.*?(http://.*?)'
            # 获取每页的所有文章链接并添加到列表 listurl 中
            listurl.append(re.compile(listurlpat, re.S).findall(data1))
        print("共获取到 " + str(len(listurl)) + " 页") # 便于调试
        return listurl
    except urllib.error.URLError as e:
        if hasattr(e, "code"):
            print(e.code)
        if hasattr(e, "reason"):
            print(e.reason)
        # 若为 URLError 异常, 延时 10 秒执行
        time.sleep(10)
    except Exception as e:
        print("exception:" + str(e))
        # 若为 Exception 异常, 延时 1 秒执行
        time.sleep(1)

# 通过文章链接获取对应内容
def getcontent(listurl, proxy):
    i = 0
    # 设置本地文件中的开始 html 编码
    html1 = '<' + '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">'
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title> 微信文章页面 </title>

```

```

</head>
<body>'
    fh=open("D:/Python35/myweb/part6/1.html","wb")
    fh.write(html1.encode("utf-8"))
    fh.close()
    # 再次以追加写入的方式打开文件,以写入对应文章内容
    fh=open("D:/Python35/myweb/part6/1.html","ab")
    # 此时 listurl 为二维列表,形如 listurl[i][j],第一维存储的信息跟几页相关,第二维存的跟该页
    # 几个文章链接相关
    for i in range(0,len(listurl)):
        for j in range(0,len(listurl[i])):
            try:
                url=listurl[i][j]
                # 处理成真实 url,读者亦可以观察对应网址的关系自行分析,采集网址比真实网址多了
                # 一串 amp
                url=url.replace("amp;","")
                # 使用代理去爬取对应网址的内容
                data=use_proxy(proxy,url)
                # 文章标题正则表达式
                titlepat="<title>(.*?)</title>"
                # 文章内容正则表达式
                contentpat='id="js_content">(.*?)id="js_sg_bar"'
                # 通过对应正则表达式找到标题并赋给列表 title
                title=re.compile(titlepat).findall(data)
                # 通过对应正则表达式找到内容并赋给列表 content
                content=re.compile(contentpat,re.S).findall(data)
                # 初始化标题与内容
                thistitle=" 此次没有获取到 "
                thiscontent=" 此次没有获取到 "
                # 如果标题列表不为空,说明找到了标题,取列表第零个元素,即此次标题赋给变量 thistitle
                if(title!=[]):
                    thistitle=title[0]
                if(content!=[]):
                    thiscontent=content[0]
                # 将标题与内容汇总赋给变量 dataall
                dataall="<p>标题为 :"+thistitle+"</p><p>内容为 :"+thiscontent+"</p><br>"
                # 将该篇文章的标题与内容的总信息写入对应文件
                fh.write(dataall.encode("utf-8"))
                print(" 第 "+str(i)+" 个网页第 "+str(j)+" 次处理 ") # 便于调试
            except urllib.error.URLError as e:
                if hasattr(e,"code"):
                    print(e.code)
                if hasattr(e,"reason"):
                    print(e.reason)
                # 若为 URLError 异常,延时 10 秒执行
                time.sleep(10)
            except Exception as e:
                print("exception:"+str(e))
                # 若为 Exception 异常,延时 1 秒执行
                time.sleep(1)

```

```

fh.close()
# 设置并写入本地文件的html 后面结束部分代码
html2='''</body>
</html>
'''

fh=open("D:/Python35/myweb/part6/1.html","ab")
fh.write(html2.encode("utf-8"))
fh.close()
# 设置关键词
key=" 物联网 "
# 设置代理服务器, 该代理服务器有可能失效, 读者需要换成新的有效代理服务器
proxy="119.6.136.122:80"
# 可以为 getlisturl() 与 getcontent() 设置不同的代理服务器, 此处没有启用该项设置
proxy2=""
# 起始页
pagestart=1
# 爬取到哪页
pageend=2
listurl=getlisturl(key,pagestart,pageend,proxy)
getcontent(listurl,proxy)

```

执行结果如下:

```

>>>
===== RESTART: D:\Python35\6.4.py =====
共获取到 2 页
第 0 个网页第 0 次处理
第 0 个网页第 1 次处理
exception:IncompleteRead(45698 bytes read, 34469 more expected)
exception:expected string or bytes-like object
第 0 个网页第 3 次处理
第 0 个网页第 4 次处理
第 0 个网页第 5 次处理
第 0 个网页第 6 次处理
第 0 个网页第 7 次处理
第 0 个网页第 8 次处理
第 0 个网页第 9 次处理
第 1 个网页第 0 次处理
第 1 个网页第 1 次处理
第 1 个网页第 2 次处理
第 1 个网页第 3 次处理
第 1 个网页第 4 次处理
第 1 个网页第 5 次处理
第 1 个网页第 6 次处理
第 1 个网页第 7 次处理
第 1 个网页第 8 次处理
第 1 个网页第 9 次处理

```

可以看到, 执行过程中可能会出现异常, 但是由于我们设置了异常处理, 所以即使出现异常, 也不会停止爬行, 延时后再次进行爬行。

爬虫爬取信息后将信息保存在本地网页“D:/Python35/myweb/part6/1.html”中，我们打开该网页，发现对应文章已经全部保存了，这里只截取了部分文章，如图6-11所示：

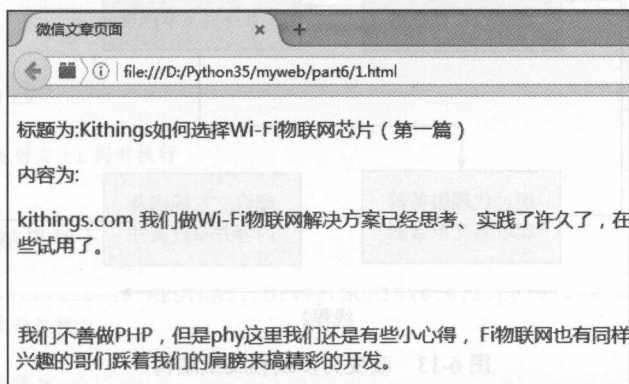


图 6-11 在本地网页中展示的爬取成功界面

此时，已经自动将网页上多篇关于“物联网”的文章爬取到了本地文件中。如果我们希望自动获取其他主题的文章，只需改变该程序中的关键词参数即可。

## 6.5 什么是多线程爬虫

之前我们所写的爬虫，这些程序在执行起来是有先后顺序的，比如，上一节中的微信爬虫核心部分的执行顺序大致如图6-12所示：

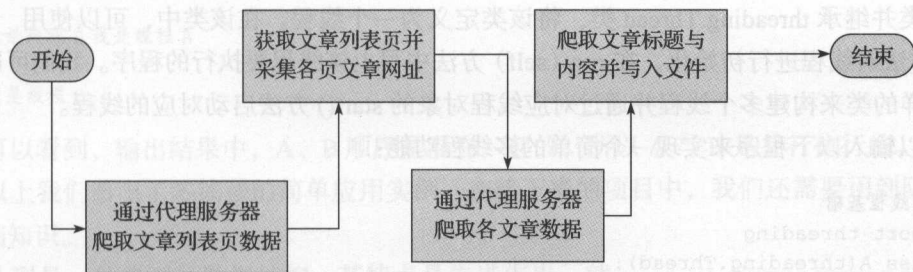


图 6-12 6.4 节微信爬虫的核心执行过程

可以看到，该爬虫的程序是依次执行的，即执行流程都在一条线上，这种执行结构称为单线程结构，对应的爬虫称为单线程爬虫。

所谓的多线程爬虫，指的是爬虫中的某部分程序可以并行执行，即在多条线上执行，这种执行结构称为多线程结构，对应的爬虫称为多线程爬虫。

比如，上面的单线程微信爬虫，在不影响程序结果的情况下，可以将执行流程整理一下，变为多线程爬虫，如可以将其执行结构改变为如图6-13形式：



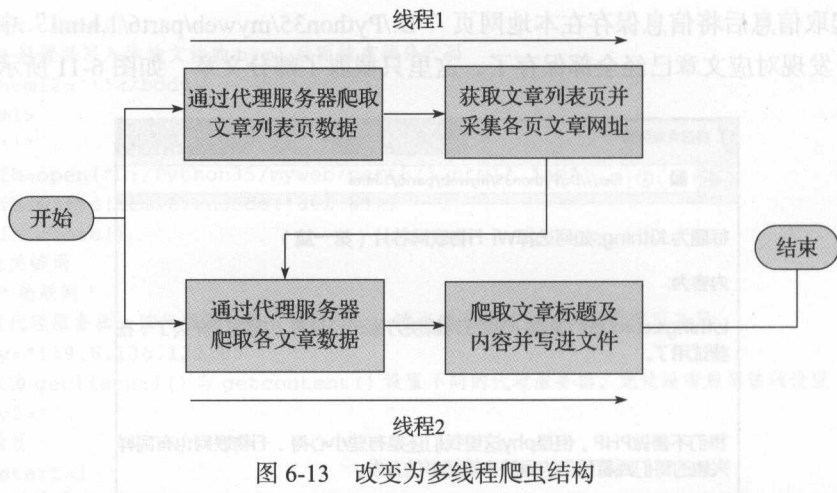


图 6-13 改变为多线程爬虫结构

可以看到，此时的程序可以通过两条线程并行执行，程序运行的效率可能会提高，因为线程 1 和线程 2 可以同时执行。关于多线程的好处还有很多，在此我们就不一一讲解，在此，我们只要学会掌握多线程的基本概念，并能够在实战中编写出多线程爬虫就可以。

## 6.6 多线程爬虫实战

在本节中，我们会讲解多线程爬虫的实战基础知识，并跟大家一起将上面的单线程微信爬虫改成一个多线程微信爬虫。

要在 Python 中使用多线程，我们可以导入 `threading` 模块使用多线程功能。我们可以定义一个类并继承 `threading.Thread` 类，将该类定义为一个线程。在该类中，可以使用 `__init__` (self) 方法对线程进行初始化，在 `run` (self) 方法中写上该线程要执行的程序。我们可以声明多个这样的类来构建多个线程并通过对对应线程对象的 `start()` 方法启动对应的线程。

可以输入以下程序来实现一个简单的多线程功能：

```

# 多线程基础
import threading
class A(threading.Thread):
    def __init__(self):
        # 初始化该线程
        threading.Thread.__init__(self)
    def run(self):
        # 该线程要执行的程序内容
        for i in range(10):
            print("我是线程 A")
class B(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):

```

```

for i in range(10):
    print("我是线程B")
# 实例化线程A为t1
t1=A()
# 启动线程t1
t1.start()
# 实例化线程B为t2
t2=B()
# 启动线程t2,此时与t1同时执行
t2.start()

```

该程序的执行结果如下:

```

===== RESTART: D:\Python35\6.6.1.py =====
>>> 我是线程A我是线程B

```

我是线程A我是线程B

我是线程A我是线程B

我是线程A我是线程B

我是线程A

我是线程B我是线程A

我是线程B我是线程A

我是线程B我是线程A

我是线程B我是线程A

我是线程B我是线程A

我是线程B

可以看到,输出结果中,A、B顺序是混在一起的,所以A与B是并行执行的。

以上我们给出了多线程的简单应用实例,在接下来的项目中,我们还需要用到队列的一些基础知识。

队列是一种典型的数据结构,其特点是先进先出,就像排队一样,谁先排进去,谁就先出来。

如图6-14所示,即为对应的队列数据结构特点描述,假如队列的方向是向下的,A先进了队列,随后B、C元素再依次进了队列,此时D元素正在进队列的过程中,那么出队列的顺序必然是A、B、C、D。即满足先进先出的规律。

队列的功能在Python中非常好实现。我们可以通过导

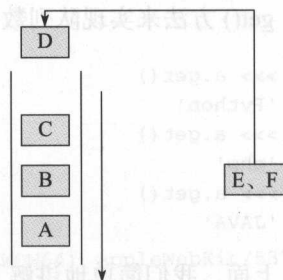


图6-14 队列数据结构特点示意图

入 queue 模块来实现对应队列的功能。导入后, 可以通过 `queue.Queue()` 实例化一个队列对象, 并且通过队列对象的 `put()` 方法实现将一个数据入队列的操作, 每次入完队列之后, 可以通过 `task_done()` 方法表示该次入队列任务完成, 如果要出队列, 则可以通过队列对象的 `get()` 方法实现。我们可以通过以下例子进行分析讲解。

首先通过如下代码导入 queue 模块。

```
>>> import queue
```

导入后可以通过以下代码创建一个队列 a, a 为一个队列对象。

```
>>> a=queue.Queue()
```

可以通过 `put(数据)`, 将对应的数据传入队列中。如下所示, 我们将字符串数据 "hello" 传递到了队列 a 中。

```
>>> a.put("hello")
```

入队列完成之后, 可以通过如下代码表示该次入队列任务完成。

```
>>> a.task_done()
```

假如此时我们想取出刚才传入的数据, 可以通过 `get()` 方法直接取出, 对应程序及执行结果如下所示:

```
>>> a.get()
'hello'
```

同样, 我们可以调用多次 `put()` 方法将多个数据依次入队列。如下所示:

```
>>> a.put("Python")
>>> a.task_done()
>>> a.put("php")
>>> a.task_done()
>>> a.put("JAVA")
>>> a.task_done()
```

如果想从队列中取出这些数据, 则会按先进先出的顺序输出对应的结果, 我们可以一次调用 `get()` 方法来实现队列数据的输出。程序及执行结果如下所示:

```
>>> a.get()
'Python'
>>> a.get()
'php'
>>> a.get()
'JAVA'
```

上面, 我们简单地讲解了队列这种数据结构, 我们在下面的爬虫编写中, 会将获得的对应文章网址依次存的队列中, 然后由另一个线程依次从该队列中获得要爬取的文章网址, 从

而爬取出对应的内容。

有了这些基础知识之后，接下来我们为大家讲解如何将上面的单线程微信网络爬虫更改为多线程微信网络爬虫。

更改的思路如下：

1) 总体规划好程序执行的流程，并规划好各线程的关系与作用。本项目中将划分为 3 个线程。

2) 线程 1 专门获取对应网址并处理为真实网址，然后将网址写入队列 `urlqueue` 中，该队列专门用来存放具体文章的网址。

3) 线程 2 与线程 1 并行执行，从线程 1 提供的文章网址中依次爬取对应文章信息并处理，处理后将我们需要的结果写入对应的本地文件中。

4) 线程 3 主要用于判断程序是否完成。因为在此若没有一个总体控制的线程，即使线程 1、2 执行完，也不会退出程序，这不是我们想要的结果，所以我们可以建立一个新的线程，专门用来实现总体控制，每次延时 60 秒，延时后且存放网址的队列 `urlqueue` 中没有了网址数据，说明线程 2 已经 GET 完全部的网址了（不考虑线程 1 首次无法将网址写入队列的特殊情况，如果爬取没问题，60 秒的时间完全足够执行完第一次爬取与写入的操作。也不考虑线程 2 爬取完网址但线程 1 尚未执行完下一次写入网址的操作的情况，因为线程 1 会比线程 2 快很多，即使线程 1 延时较长时间等待线程 2 的执行，正常情况下，线程 1 速度仍会比线程 2 快。），即此时已经爬取完所有的文章信息，所以此时可以由线程 3 控制退出程序。

5) 在正规的项目设计的时候，我们会希望并行执行的线程执行的时间相近，因为这样整个程序才能达到更好的平衡，如果并行执行的线程执行时间相差较大，会发生某一个线程早早执行完成，而另一些线程迟迟未完成的情况，这样显然程序不够平衡，自然效率以及线程设计有待改进。从这一点来说，本项目仍然有完善的空间。

6) 建立合理的延时机制，比如在发生异常之后，进行相应的延时处理。再比如也可以通过延时机制让执行较快的线程进行延时，等待一下执行较慢的线程。

7) 建立合理的异常处理机制。

多线程微信网络爬虫的完整代码如下，核心部分均有详细注释：

```
import threading
import queue
import re
import urllib.request
import time
import urllib.error

urlqueue=queue.Queue()
# 模拟成浏览器
headers=("User-Agent","Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0")
opener = urllib.request.build_opener()
```



```

opener.addheaders = [headers]
# 将 opener 安装为全局
urllib.request.install_opener(opener)
listurl=[]
# 使用代理服务器的函数
def use_proxy(proxy_addr,url):
    try:
        proxy= urllib.request.ProxyHandler({'http':proxy_addr})
        opener = urllib.request.build_opener(proxy, urllib.request.HTTPHandler)
        urllib.request.install_opener(opener)
        data = urllib.request.urlopen(url).read().decode('utf-8')
        return data
    except urllib.error.URLError as e:
        if hasattr(e,"code"):
            print(e.code)
        if hasattr(e,"reason"):
            print(e.reason)
            time.sleep(10)
    except Exception as e:
        print("exception:"+str(e))
        time.sleep(1)
# 线程 1, 专门获取对应网址并处理为真实网址
class geturl(threading.Thread):
    def __init__(self,key,pagestart,pageend,proxy,urlqueue):
        threading.Thread.__init__(self)
        self.pagestart=pagestart
        self.pageend=pageend
        self.proxy=proxy
        self.urlqueue=urlqueue
        self.key=key
    def run(self):
        page=self.pagestart
        # 编码关键词 key
        keycode=urllib.request.quote(key)
        # 编码 "&page"
        pagecode=urllib.request.quote("&page")
        for page in range(self.pagestart,self.pageend+1):
            url="http://weixin.sogou.com/weixin?type=2&query="+keycode+pagecode+str(page)
            # 用代理服务器爬取, 解决 IP 被封杀问题
            data1=use_proxy(self.proxy,url)
            # 列表页 url 正则
            listurlpat='<div class="txt-box">.*?(http://.*?)'
            listurl.append(re.compile(listurlpat,re.S).findall(data1))
# 便于调试
print(" 获取到 "+str(len(listurl))+" 页 ")
for i in range(0,len(listurl)):
    # 等一等线程 2, 合理分配资源
    time.sleep(7)
    for j in range(0,len(listurl[i])):
        try:

```

```

url=listurl[i][j]
# 处理成真实 url, 读者亦可以观察对应网址的关系自行分析, 采集网址比真实网
址多了一串 amp
url=url.replace("amp;", "")
print(" 第 "+str(i)+"i"+str(j)+"j 次入队 ")
self.urlqueue.put(url)
self.urlqueue.task_done()
except urllib.error.URLError as e:
    if hasattr(e, "code"):
        print(e.code)
    if hasattr(e, "reason"):
        print(e.reason)
    time.sleep(10)
except Exception as e:
    print("exception:"+str(e))
    time.sleep(1)

```

# 线程 2, 与线程 1 并行执行, 从线程 1 提供的文章网址中依次爬取对应文章信息并处理

class getcontent(threading.Thread):

```

    def __init__(self, urlqueue, proxy):

```

```

        threading.Thread.__init__(self)

```

```

        self.urlqueue=urlqueue

```

```

        self.proxy=proxy

```

```

    def run(self):

```

```

        html1='<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

```

```

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

```

```

<html xmlns="http://www.w3.org/1999/xhtml">

```

```

<head>

```

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

```

```

<title> 微信文章页面 </title>

```

```

</head>

```

```

<body>'

```

```

        fh=open("D:/Python35/myweb/part6/2.html", "wb")

```

```

        fh.write(html1.encode("utf-8"))

```

```

        fh.close()

```

```

        fh=open("D:/Python35/myweb/part6/2.html", "ab")

```

```

        i=1

```

```

        while(True):

```

```

            try:

```

```

                url=self.urlqueue.get()

```

```

                data=use_proxy(self.proxy, url)

```

```

                titlepat="<title>(.*?)</title>"

```

```

                contentpat='id="js_content">(.*?)id="js_sg_bar"'

```

```

                title=re.compile(titlepat).findall(data)

```

```

                content=re.compile(contentpat, re.S).findall(data)

```

```

                thistitle=" 此次没有获取到 "

```

```

                thiscontent=" 此次没有获取到 "

```

```

                if(title!=[]):

```

```

                    thistitle=title[0]

```

```

                if(content!=[]):

```

```

                    thiscontent=content[0]

```

```

                dataall="<p> 标题为: "+thistitle+"</p><p> 内容为: "+thiscontent+"</p><br>"

```

```

        fh.write(dataall.encode("utf-8"))
        print(" 第 "+str(i)+" 个网页处理 ") # 便于调试
        i+=1
    except urllib.error.URLError as e:
        if hasattr(e,"code"):
            print(e.code)
        if hasattr(e,"reason"):
            print(e.reason)
            time.sleep(10)
    except Exception as e:
        print("exception:"+str(e))
        time.sleep(1)
    fh.close()
    html2=''</body>
</html>
'''
    fh=open("D:/Python35/myweb/part6/2.html","ab")
    fh.write(html2.encode("utf-8"))
    fh.close()
# 并行控制程序,若60秒未响应,并且存url的队列已空,则判断为执行成功
class conrl(threading.Thread):
    def __init__(self,urlqueue):
        threading.Thread.__init__(self)
        self.urlqueue=urlqueue
    def run(self):
        while(True):
            print(" 程序执行中 ")
            time.sleep(60)
            if(self.urlqueue.empty()):
                print(" 程序执行完毕! ")
                exit()

key=" 人工智能 "
proxy="119.6.136.122:80"
proxy2=""
pagestart=1# 起始页
pageend=2# 爬取到哪页
# 创建线程1对象,随后启动线程1
t1=geturl(key,pagestart,pageend,proxy,urlqueue)
t1.start()
# 创建线程2对象,随后启动线程2
t2=getcontent(urlqueue,proxy)
t2.start()
# 创建线程3对象,随后启动线程3
t3=conrl(urlqueue)
t3.start()

```

程序执行结果如下所示:

```

>>>
===== RESTART: D:\Python35\6.6.2.py =====
程序执行中

```

```
>>>
```

```
获取到 2 页
```

```
第 0i0j 次入队
```

```
第 0i1j 次入队
```

```
第 0i2j 次入队
```

```
第 0i3j 次入队
```

```
第 0i4j 次入队
```

```
第 0i5j 次入队
```

```
第 0i6j 次入队
```

```
第 0i7j 次入队
```

```
第 0i8j 次入队
```

```
第 0i9j 次入队
```

```
第 1 个网页处理
```

```
第 2 个网页处理
```

```
第 1i0j 次入队
```

```
第 1i1j 次入队
```

```
.....
```

由于篇幅有限，我们只展示前面部分的输出结果，可以看到，线程 1 与线程 2 是并行执行的，并且每隔 60 秒钟会由线程 3 判断程序是否执行完成，若执行完成，则退出该程序，若未执行完成，则继续由线程 1 与线程 2 并行执行。

此时，用浏览器打开对应程序写入的本地文件“D:/Python35/myweb/part6/2.html”，本地网页的内容如图 6-15 所示：

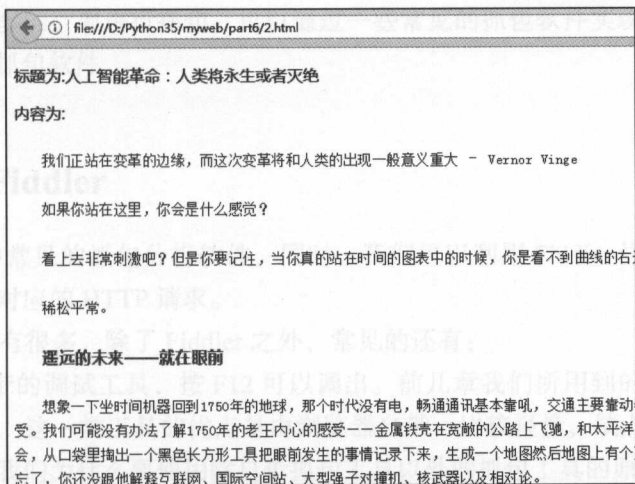


图 6-15 本地网页中展示成功爬取到的内容

可以看到，此时已经自动爬取了与“人工智能”相关的文章。由于篇幅有限，截图为爬取网页的部分界面。

在此，我们成功通过多线程技术改造了 Python 微信网络爬虫，相信大家通过这个例子，可以更好地了解并掌握多线程爬虫。



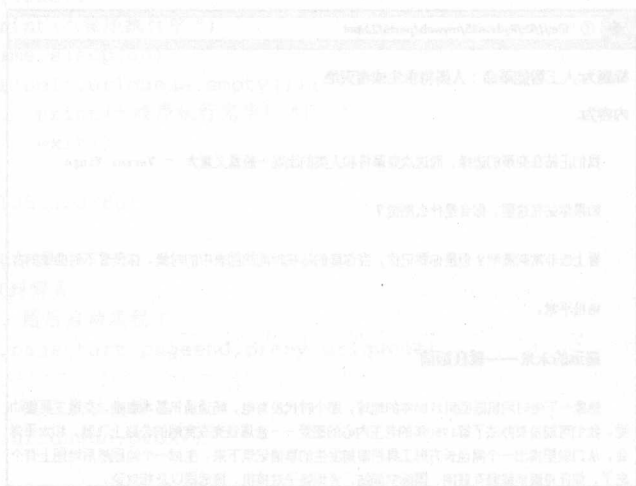
## 6.7 小结

1) 进行网页信息提取分析的时候,经常要学会寻找特殊标识,特殊标识要满足唯一性,并且包含要爬取的信息,以及尽量少的无关信息。

2) 通过爬虫进行自动化爬取,可以为我们省下很多事情。比如,有些站长需要采集一些内容到自己的网站上,如果通过复制粘贴的方式,耗费的精力非常大,而采用爬虫的方式,我们可以直接将关注的数据爬取下来,并可以用程序直接自动写进对应的数据库中,此时,网站上的内容就可以实现自动更新了。当然,学习爬虫需要遵守法律,比如一些别人原创性的内容,我们应当声明转载,尊重别人的原创成果,对于一些禁止爬取或禁止转载的内容,我们亦应当遵守相关规定。

3) 我们在爬取微信文章的时候,经常会被官方屏蔽 IP,这个问题我们可以采用 4.6 节提到的代理服务的方式解决,同样,我们可以在 <http://yum.iqianyue.com/proxy> 中获取到最新的代理服务器及端口,并尝试用这些代理服务器爬取对应网页,当然有些新代理服务器也可能失效,在这种情况下可以多试几个,或者通过互联网寻找一些稳定的更新迅速的代理服务器地址。

4) 所谓的多线程爬虫,指的是爬虫中的某部分程序可以并行执行,即在多条线上执行,这种执行结构称为多线程结构,对应的爬虫称为多线程爬虫。



## 学会使用 Fiddler

我们使用计算机上的浏览器或者客户端软件要与外界进行通信，就必然会有数据的发送或接收，有的时候，我们需要对这些传递的数据进行分析，就需要截获这些传递的数据，其中对这些数据进行截获、重发、编辑、转存的过程叫作抓包。在写爬虫的时候，抓包分析用得相对来说也是较多的，要进行抓包，可以通过一些常见的抓包软件实现，Fiddler 就是一种常见的比较好用的抓包软件。

### 7.1 什么是 Fiddler

Fiddler 是一种常见的抓包分析软件，同时，我们可以利用 Fiddler 详细地对 HTTP 请求进行分析，并模拟对应的 HTTP 请求。

目前抓包软件有很多，除了 Fiddler 之外，常见的还有：

1) 浏览器自带的调试工具，按 F12 可以调出。前几章我们所用到的抓包工具就是浏览器自带的调试工具。这一类工具的优点是由浏览器自带，比较轻量，缺点是不能支持一些复杂的抓包，这也是我们为什么要使用除自带抓包工具以外的抓包工具的原因。

2) Wireshark，这是一款通用的抓包工具，功能比较齐全，正因为其功能比较齐全，所以较为庞大，而我们写爬虫的时候主要是分析 HTTP 请求，所以这款软件的很多功能都用不到，故而我们没有选择介绍这款软件。

我们知道，写爬虫的时候配合 Fiddler 这款抓包软件来使用是比较合适的，故而本章中会为大家讲解 Fiddler 的常见功能的使用。

## 7.2 爬虫与 Fiddler 的关系

有人可能会问，Fiddler 既然是一款抓包分析软件，那么网络爬虫与 Fiddler 到底有什么关系？

网络爬虫是自动爬取网页的程序，在爬取的过程中必然涉及客户端与服务器端之间的通信，自然也需要发送一些 HTTP 请求，并接收服务器返回的结果。在一些稍复杂的网络请求中，我们直接看网址的变化是看不出规律的，此时如果要进行自动化爬取网页，就必须要通过程序构造这些请求，而要通过程序构造这些请求，就必须首先分析这些请求的规律。所以此时我们要使用工具截获这些请求，对这些请求进行分析，这个过程如果使用抓包软件配合进行，则将会变得更加方便。

比如，我们在浏览一些网页时，浏览到最下面的时候会出现一个“加载更多”的字样，此时单击“加载更多”则会展现出更多的内容，加载出来的内容跟原内容是在同一个网页上展示的。我们直接看网址的变化看不出任何规律，便无法分析该请求是如何实现的，自然也就无法通过程序构造出该请求，因此只能通过手动单击实现“加载更多”，这显然不是我们所希望的结果，那么怎样解决这个问题呢？

此时可以使用 Fiddler 进行抓包，并对这些数据进行分析，这样就可以分析出实现“加载更多”功能的实现方法，知道其实现规律后，就可以通过编写程序构造出对应的请求，并由程序自动地实现这些请求的发送。

同样，在进行登录的时候，很多网页的真实登录处理地址并不是我们看到的网址，这些网址一般需要通过工具进行分析得出，比如之前我们在实现登录功能的时候，就是通过浏览器自带的调试工具来分析真正的登录处理网址的，同样，我们也可以使用 Fiddler 分析出真实登录处理网址，在学习 Fiddler 之后，我们更多地会以 Fiddler 为例来讲解对应的网络数据包如何分析。

所以可以看到，编写爬虫的时候，不一定会都会用到 Fiddler，但对某些稍复杂网页进行爬取的时候，利用 Fiddler 可以更好、更快、更方便地分析对应网页，从而编写出对应爬虫。

## 7.3 Fiddler 的基本原理与基本界面

在学习使用 Fiddler 之前，我们首先需要对 Fiddler 的基本原理以及基本界面进行简单的了解。

Fiddler 是如何工作的呢？为什么 Fiddler 可以实现抓包的功能呢？

首先我们对 Fiddler 的基本工作原理进行简单的讲解。

如果没有 Fiddler，本地的客户端软件或浏览器与互联网服务器之间的通信可以简化为如图 7-1 所示。

有了 Fiddler 之后,其通信过程如图 7-2 所示。

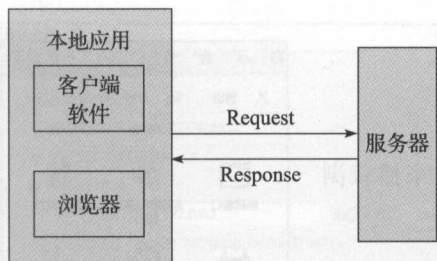


图 7-1 本地应用与服务之间的通信

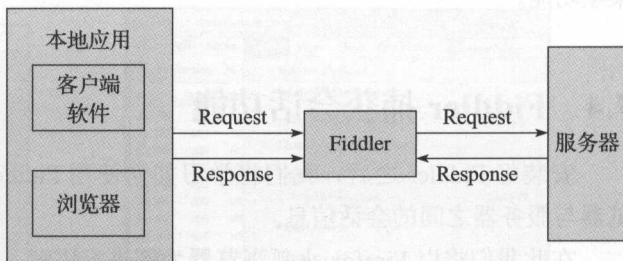


图 7-2 Fiddler 基本工作原理

Fiddler 的基本工作原理可以通过图 7-2 形象表示,可以看到,如果没有 Fiddler,本地应用如果要与服务器进行通信,可以直接向服务器发送 Request 请求,待服务器处理之后将处理结果返回本地,本地应用接收响应 Response。

如果有了 Fiddler,本地应用与服务器之间所有的 Request 和 Response 都将经过 Fiddler,由 Fiddler 进行转发,可以看到,此时 Fiddler 以代理服务的方式存在。由于所有的网络数据都会经过 Fiddler,自然 Fiddler 能够截获这些数据,实现网络数据的抓包。

在对 Fiddler 的原理进行了简单的了解之后,接下来我们将进一步认识 Fiddler。

要使用 Fiddler,首先需要安装 Fiddler 这款软件。我们可以从 Fiddler 的官网 (<http://www.telerik.com/fiddler>) 下载 Fiddler,下载之后打开直接安装即可。

安装之后,我们就可以打开 Fiddler 这款软件了。打开之后,基本界面如图 7-3 所示:

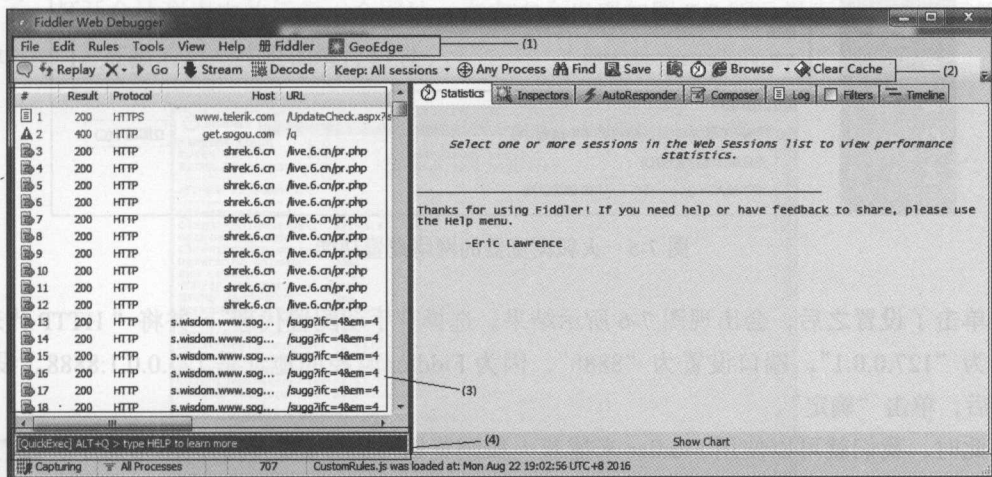


图 7-3 Fiddler 基本界面

图中(1)处所指位置为 Fiddler 的菜单栏,(2)处所指的位置为 Fiddler 的工具栏,(3)处所指的位置为 Fiddler 的会话列表,我们本地应用与互联网通信的会话信息在此显示,(4)处所



指位置为 Fiddler 的 QuickExec 命令输入窗口，在此我们可以输入一些 Fiddler 指令来快速实现某项功能。

## 7.4 Fiddler 捕获会话功能

安装好 Fiddler 之后，我们将学习如何使用 Fiddler 来捕获浏览器与服务器之间的会话信息。

在此我们将以 Firefox 火狐浏览器为例进行讲解。

因为 Fiddler 是以代理服务器的方式进行工作的，所以我们首先应当设置火狐浏览器，让火狐浏览器使用 Fiddler 作为其代理服务器。

设置火狐浏览器的方法如下。

首先单击“更多”中的“选项”，如图 7-4 所示。

随后，在弹出的界面中单击“高级”，在“高级”中将标签切换为“网络”，随后在“网络”下方可以看到“连接”的字样。我们单击“连接”右方的“设置”，如图 7-5 所示。



图 7-4 火狐浏览器的设置

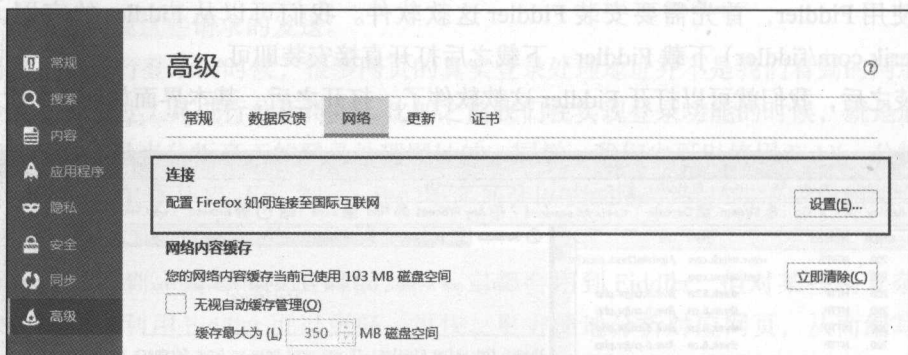


图 7-5 火狐浏览器的网络设置功能

单击了设置之后，会出现图 7-6 所示结果。选择“手动配置代理”，并将“HTTP 代理”设置为“127.0.0.1”，端口设置为“8888”，因为 Fiddler 监控的地址是 127.0.0.1:8888。设置好之后，单击“确定”。

此时，我们就可以使用 Fiddler 来捕获火狐浏览器与服务器之间的会话信息了。我们知道，现在的网站有的使用 HTTP 协议，有的使用的是 HTTPS 协议。

如果想让 Fiddler 能够捕获 HTTPS 的会话信息，还需要设置一下 Fiddler。

打开 Fiddler，然后单击“Tools → Fiddler Options”，如图 7-7 所示。

随后，在弹出来的界面中选择“HTTPS”标签，将下方选项全部勾选上，如图 7-8

所示。

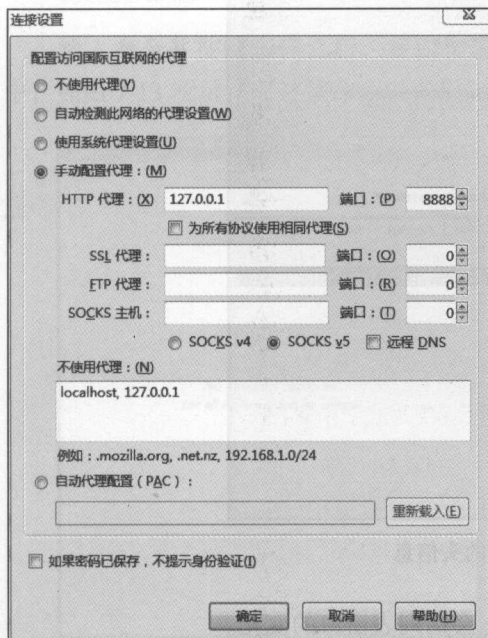


图 7-6 火狐浏览器的代理设置

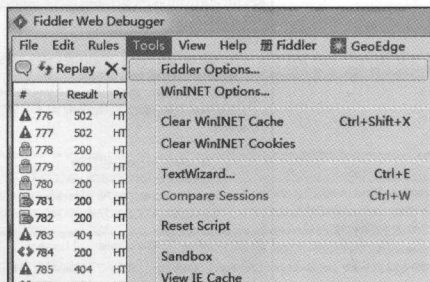


图 7-7 Fiddler 中的 Options 设置

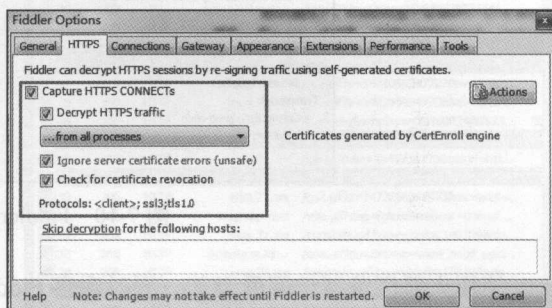


图 7-8 Fiddler 中的 HTTPS 设置

配置好之后，现在 Fiddler 就能捕获火狐浏览器与服务器之间的 HTTP 和 HTTPS 会话信息了。打开会话列表中的任意一个网址，在右方会出现如图 7-9 所示界面，此时的标签为“Statistics”，显示的是一些页面统计信息。

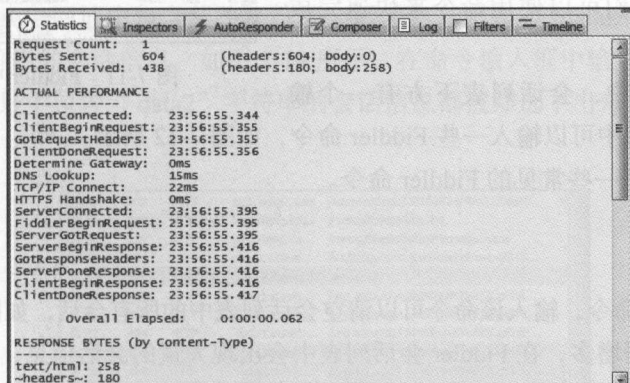


图 7-9 Fiddler 中的页面统计信息

将标签切换为“Inspectors”，显示的是一些嗅探信息，并且该标签下有很多子标签，比如“Headers”子标签表示的是网页的一些头信息，如图 7-10 所示。

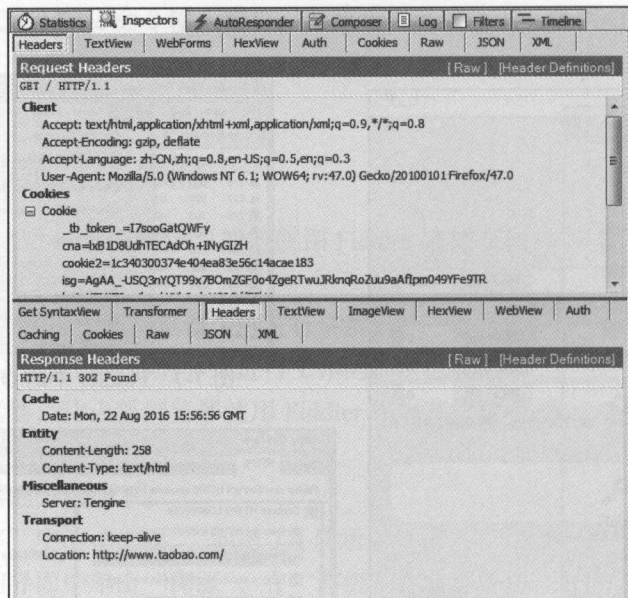


图 7-10 Fiddler 中的头信息

使用火狐浏览器访问爱奇艺，同时观察 Fiddler 会话列表窗口的变化。

可以看到，此时 Fiddler 捕获了很多访问爱奇艺网站时的会话信息，如图 7-11 所示。

## 7.5 使用 QuickExec 命令行

在 Fiddler 中我们可以使用命令来快速完成一些功能。

在 Fiddler 界面中，会话列表下方有一个输入框，在这个输入框中可以输入一些 Fiddler 命令，如图 7-12 所示。

在此，我们讲解一些常见的 Fiddler 命令。

### 1. cls

cls 命令为清屏命令，输入该命令可以清空会话列表中的所有会话，如图 7-12 所示。有时候，由于传递的数据增多，在 Fiddler 会话列表中会出现大量的会话信息，此时界面相对来说较为杂乱，不便于我们对新会话信息进行分析，所以可以输入该命令清空会话信息列表。

### 2. select

通过 select 命令我们可以选择出某一类型 HTTP 会话的功能，比如想选择出所有的 html

No.	Local Address	Remote Address	Protocol	URL	Size
6	200	HTTP	www.iqiyi.com	/player/cupid/common/de...	31,392
7	200	HTTP	nsdick.baidu.com	/v.gif?pid=324&iqiyi_cooki...	0
8	200	HTTP	cpo.baidu.com	/cpo/ia/html/sync.htm?...	0
9	200	HTTP	iqiyi.lrs01.com	/ir17_jwt_id=8_jwt_UA=U...	43
10	200	HTTP	hnu.baidu.com	/res.gif?cc=0&dc=1&dc=2...	43
11	200	HTTP	hnu.baidu.com	/res.gif?cc=0&dc=1&dc=2...	43
12	200	HTTP	msg.71.am	/cp2.gif?ts=14720520493...	0
13	200	HTTP	t7z.cupid.iqiyi.com	/show/22cb=jQuery01924...	8,360
14	200	HTTP	nl.notice.iqiyi.com	/api/msg/hasnew.action?...	112
15	200	HTTP	search.video.qiyi.c...	/m?if=defaultQuery&resp...	522
16	200	HTTP	iplocation.geo.qiyi.c...	/cityjson?m=1546146811...	255
17	200	HTTP	www.iqiyi.com	/player/cupid/common/de...	31,392
18	200	HTTP	msg.71.am	/cp2.gif?ts=14720520493...	0
19	200	HTTP	msg.71.am	/cp2.gif?ts=14720520493...	0

图 7-11 Fiddler 中的会话信息

网页类型的 HTTP 会话，可以输入命令：

```
select html
```

输入命令并按回车键之后，我们会发现，在会话列表中所有 html 类型的会话都已被选中，如图 7-13 所示，通过该命令，我们可以准确地选出某一类型的会话。

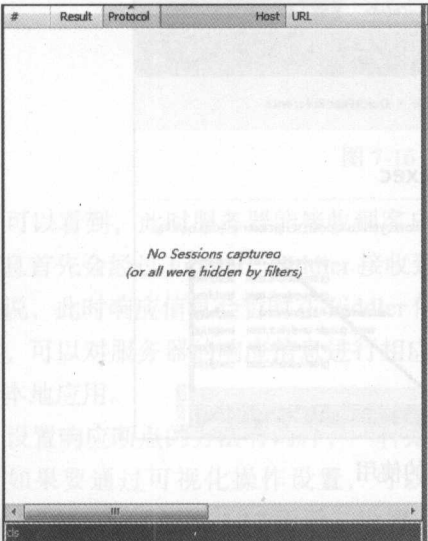


图 7-12 Fiddler 中的命令输入界面

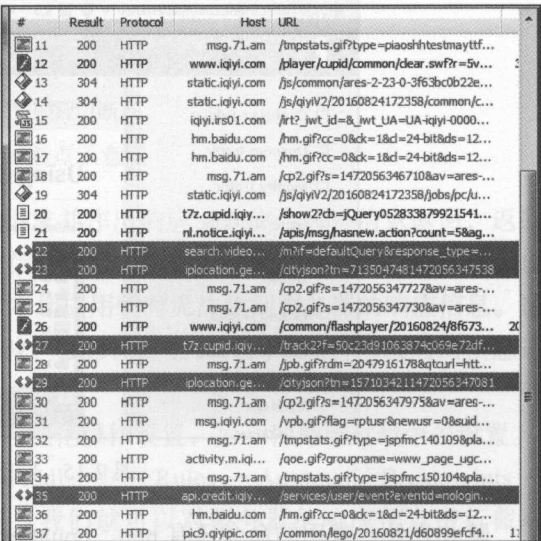


图 7-13 select 命令的使用

如果想选择出所有图片类型的会话信息，则可以输入 select image 并按回车键执行。

3. ?

? 命令可以查找出网址中包含某些字符的会话信息，比如“?data2”可以查找出网址中包含“data2”字符串的会话信息，如图 7-14 所示，在命令输入框中输入“?data2”，此时在会话列表中所有网址中包含“data2”字符串的会话信息都被筛选了出来，跟其他会话信息用不同颜色区分。

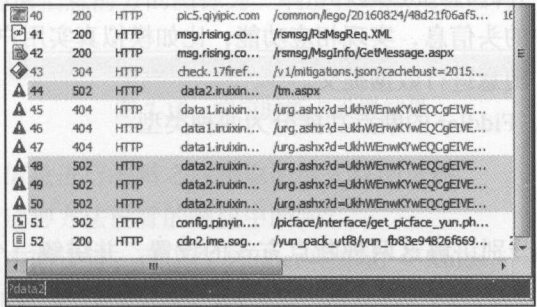


图 7-14 ? 命令的使用



#### 4. help

执行该命令可以打开 Fiddler 官方的使用手册，比如我们输入 help 命令，则会自动打开官方的使用手册，如图 7-15 所示。

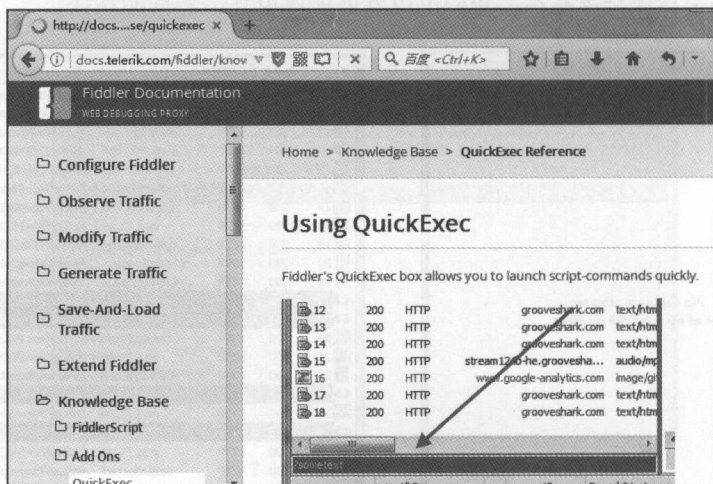


图 7-15 help 命令的使用

除此之外，常用的命令还有 bpu、bpuafter，这两个命令通常用于设置断点，我们将会在下节进行详细讲解。

## 7.6 Fiddler 断点功能

Fiddler 是以作为代理服务器的方式进行工作的，所以，本地应用与服务器传递的这些数据都会经过 Fiddler，有的时候，我们希望在传递的中间进行修改后再传递，那么可以使用 Fiddler 的断点功能。

使用 Fiddler 的断点功能，我们可以实现：

- 1) 拦截响应数据，并进行相应修改。
- 2) 修改请求数据中的头信息，实现相应功能，比如模拟真实用户请求等功能。
- 3) 构建请求数据，随意进行数据提交。

由此我们可以看到，Fiddler 的断点功能分为两种类型：

- 1) 响应时断点。
- 2) 请求时断点。

接下来我们为大家分别讲解这两种断点方式的设置，并讲解一个响应时断点应用的小实例。

响应时断点的断点处如图 7-16 所示：

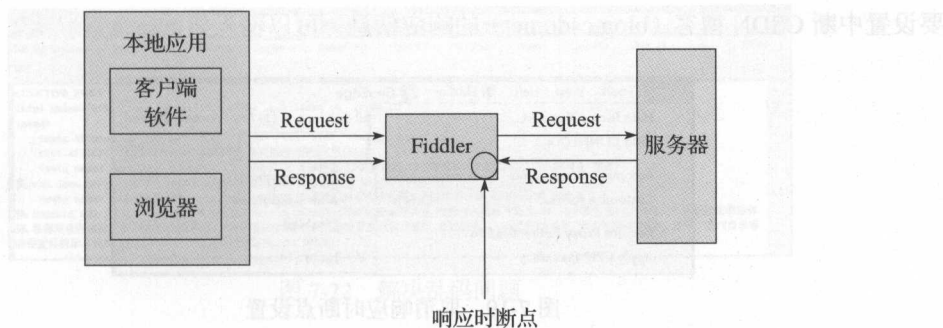


图 7-16 响应时断点示意图

可以看到，此时服务器能接收到客户端的请求并作出响应，响应之后将信息返回，返回的信息首先会经过 Fiddler，Fiddler 接收到反馈信息之后，将数据截获并中断信息的传递，也就是说，此时响应信息会暂时在 Fiddler 停留，本地应用暂时无法收到服务器的响应信息。这时候，可以对服务器的响应信息进行相应的修改，在修改之后，我们可以将修改后的信息返回给本地应用。

设置响应断点的方法有两种，一种是通过可视化操作设置，另一种是通过命令去设置。

如果要通过可视化操作设置，可以单击 Fiddler 中的 Rules → Automatic Breakpoints → After Responses，如图 7-17 所示。设置好之后，我们就可以对服务器的响应信息进行截取了。这种设置方法会中断所有网址的响应信息。

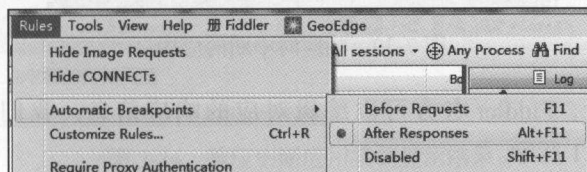


图 7-17 响应时断点的设置

我们使用这种设置方法后，通过火狐浏览器访问爱奇艺官网，就会发现此时在 Fiddler 中，www.iqiyi.com 的会话信息前方的图标为“响应在断点处被暂停”状态，如图 7-18 所示。

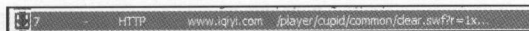


图 7-18 响应时断点设置结果

此时，可以对响应信息进行编辑之后再返回给火狐浏览器。

那么怎么取消使用这种方法设置的响应中断呢？

要取消对应的响应中断设置，可以在 Fiddler 中单击“Rules → Automatic Breakpoints → Disabled”，如图 7-19 所示。

除了使用这种可视化的方法设置响应中断外，还可以通过 bpuafter 命令设置，比如如果

要设置中断 CSDN 博客 (blog.csdn.net) 的响应信息, 可以输入如下命令:

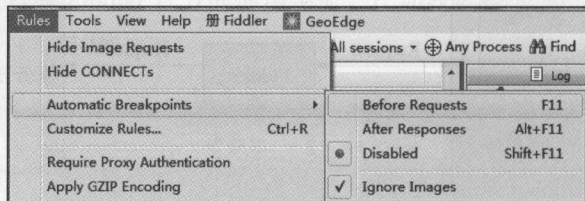


图 7-19 取消响应时断点设置

```
bpuafter blog.csdn.net
```

使用这种方法设置的响应中断, 只会中断 CSDN 博客网站相应的响应, 而对于其他网站来说, 是不会发生响应中断的。

通过这种方式设置的响应中断可以通过以下命令取消:

```
bpuafter
```

接下来讲解一个简单的响应中断应用的实例。

首先, 通过 “Rules → Automatic Breakpoints → After Responses” 设置好响应中断, 然后在火狐浏览器中访问 51CTO 学院官网 (edu.51cto.com), 然后可以发现在 Fiddler 的会话列表中对应网址的响应被中断, 如图 7-20 所示:



图 7-20 响应时中断的应用

单击该网址之后, Fiddler 右方可以编辑对应的响应信息, 我们可以将标签切换为 “TextView”, 如图 7-21 所示。

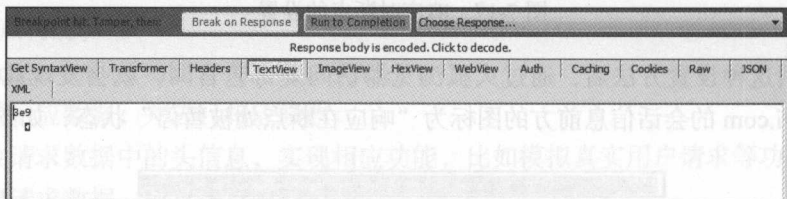


图 7-21 响应中断后设置 TextView 标签信息

可以看到, 此时的响应信息为乱码, 可以单击界面中的 “Responsebody is encoded.Click to decode.”, 单击后我们发现对应的相应信息已显示正常, 如图 7-22 所示。

随后, 可以在该编辑框中输入要返回给火狐浏览器的响应信息, 这些信息可以自定义输入。如图 7-23 所示, 输入好自定义的响应信息之后, 可以单击 “Run to Completion”, 此时会将对应的响应信息返回给火狐浏览器。

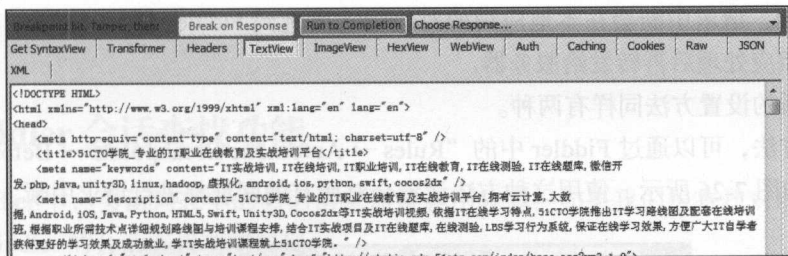


图 7-22 解决乱码问题

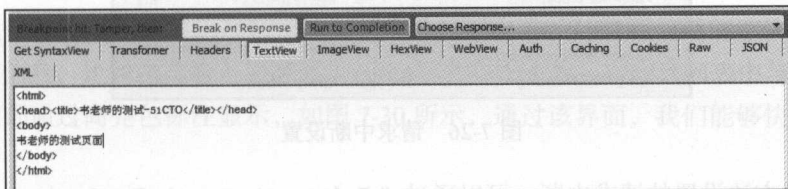


图 7-23 设置自定义响应信息

在火狐浏览器上可以看到，我们访问的是 51CTO 学院官网，但是界面却是“韦老师的测试页面”等字样，如图 7-24 所示。因为在这个过程中 51CTO 学院正常做出了响应，但是响应信息经过 Fiddler 的时候被中断，通过 Fiddler 进行修改后才返回给火狐浏览器。

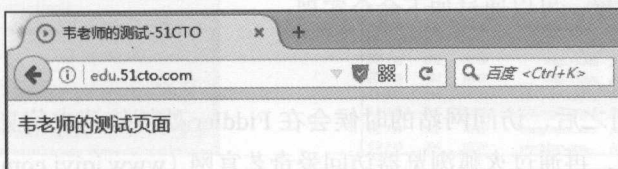


图 7-24 响应中断应用案例显示结果

这就是响应中断的一个简单的应用实例，通过响应中断我们可以让响应信息在经过 Fiddler 的时候暂时停止，进行相应的处理之后再返回给本地应用。

接下来讲解请求中断，请求中断的断点处如图 7-25 所示：

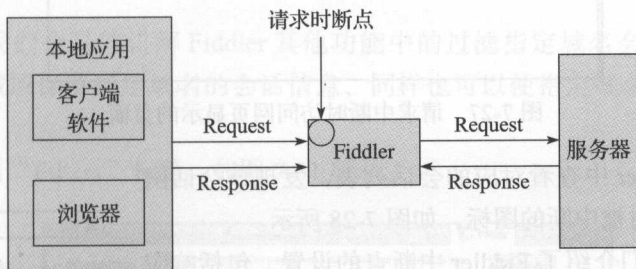


图 7-25 请求时断点示意图



可以看到,请求中断会在本地应用的对应请求发送到 Fiddler 时暂时停止,同样可以由 Fiddler 进行相应处理后再转发给服务器。

请求中断的设置方法同样有两种。

第一种方法,可以通过 Fiddler 中的“Rules → Automatic Breakpoints → Before Requests”进行设置,如图 7-26 所示,使用这种方法设置会对所有的网址都进行请求中断。

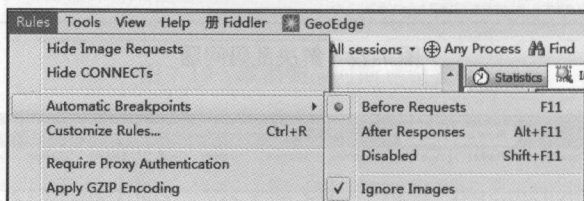


图 7-26 请求中断设置

使用这种方法设置的请求中断,可以通过“Rules → Automatic Breakpoints → Disabled”进行取消设置。

第二种方法,可以使用如下命令设置请求中断:

bpu 网址 a

通过这种方式设置的请求中断,只会对设置的网址起作用,对其他网址不起作用。如果要取消请求中断的设置,可以通过如下命令实现:

bpu

设置了请求中断之后,访问网站的时候会在 Fiddler 处暂停请求信息的传递。比如,在设置了请求中断之后,再通过火狐浏览器访问爱奇艺官网(www.iqiyi.com),会发现此时网页一直在加载中,并且显示不出任何内容,如图 7-27 所示。

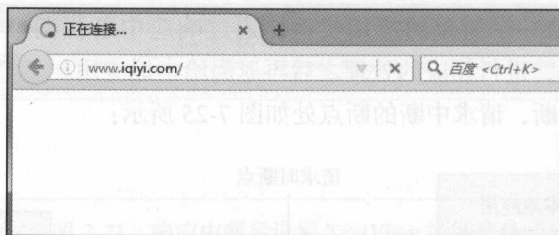


图 7-27 请求中断时访问网页显示的页面

我们再去 Fiddler 中查看对应的会话列表,发现该会话网址前面出现了请求时被中断的图标,如图 7-28 所示。

在本节中,我们介绍了 Fiddler 中断点的设置,包括响应中断和请求中断两种方式,并且为大家讲解了一个简单的断点

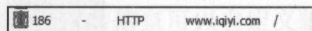


图 7-28 请求中断时 Fiddler 中对应的会话显示

应用的实例。

## 7.7 Fiddler 会话查找功能

在 Fiddler 中, 会话列表中的会话信息通常会比较多, 如果我们想查找对应的会话信息, 可以通过 Fiddler 的会话查找功能实现。

可以通过 Ctrl+F 键调出 Fiddler 的会话查找界面, 如图 7-29 所示:

此时, 可以在 Find 输入窗口中输入要查找的对应关键词, 然后单击“Find Sessions”, 相关的会话就会在会话列表中通过指定的高亮颜色显示出来。此时我们就能够快速找到这些相关的会话信息, 比如在此输入关键词“cupid”并单击查找, 在会话列表中与该关键词相关的会话信息都通过高亮色标注显示, 如图 7-30 所示, 通过该界面, 我们能够快速地找到相关信息。

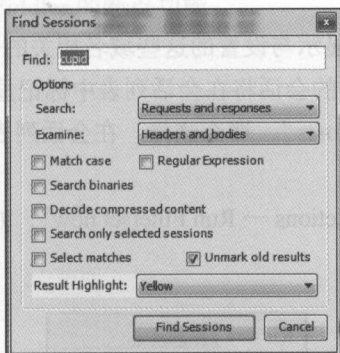


图 7-29 Fiddler 中的会话查找界面

#	Result	Protocol	Host	URL
721	304	HTTP	hm.baidu.com	/hm.js?53b7374653c37483e5dd97d78...
722	200	HTTP	static.iqiyi.com	/js/qiyiV2AugIndex_ver.js?aoqwkf
723	200	HTTP	iqiyi.sns01.com	/t?jwt_id=8_jwt_UA=UA-iqiyi-0000...
724	204	HTTP	b.scorecardre...	/b?c1=2&c2=7290408&ns_t=14720...
725	200	HTTP	msg.71.am	/mpstats.gif?type=placohdtestmayttf...
726	200	HTTP	www.iqiyi.com	/player/cupid/common/clear.swf?n=10...
727	200	HTTP	hm.baidu.com	/hm.gif?cc=0&cd=1&cd=24-bit&ds=12...
728	200	HTTP	hm.baidu.com	/hm.gif?cc=0&cd=1&cd=24-bit&ds=12...
729	200	HTTP	msg.71.am	/cp2.gif?ts=1472068175700&av=ares...
730	200	HTTP	t7z.cupid.iqiyi...	/show2?cb=jQuery033122231681620...
731	200	HTTP	nl.notice.iqiyi...	/apis/msg/haanew.action?count=5&ag...
732	200	HTTP	search.video...	/m?if=defaultQuery&response_type=...
733	200	HTTP	iplocation.geo...	/cityjson?tn=3935602951472068176057
734	200	HTTP	msg.iqiyi.com	/rpb.gif?flag=rptusr&newusr=0&aud...
735	200	HTTP	msg.71.am	/cp2.gif?ts=1472068176193&av=ares...
736	200	HTTP	msg.71.am	/cp2.gif?ts=1472068176193&av=ares...
737	200	HTTP	t7z.cupid.iqiyi...	/track2?f=50c23d91063874c069e72df...
738	200	HTTP	t7z.cupid.iqiyi...	/track2?f=50c23d91063874c069e72df...
739	302	HTTP	tyaqy.m.cn.m...	/x?k=2026208&p=71UA1&dx=0&t=2...
740	200	HTTP	iplocation.geo...	/cityjson?tn=3727365611472068175909

图 7-30 查找结果

## 7.8 Fiddler 的其他功能

同样, 使用 Fiddler 还能够实现很多其他的功能, 比如自动响应 (AutoResponder)、会话过滤等。

在本节中, 我们会具体讲解 Fiddler 其他功能中的过滤指定域名会话信息的功能。使用该功能可以隐藏或保留指定域名的会话信息, 同样也可以使指定域名的会话信息标记显示等。

首先, 切换到“Filters”标签, 如图 7-31 所示。

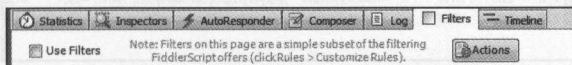


图 7-31 Filters 标签的切换

随后，勾选上“Use Filters”，并在文本框中输入对应要过滤的域名，如果是多个域名，域名间通过逗号隔开，如图 7-32 所示：

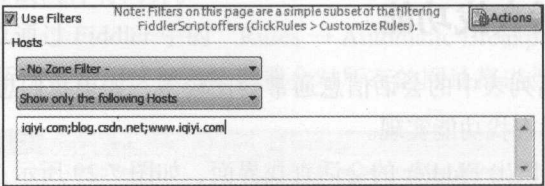


图 7-32 过滤域名的设置

随后，需要设置 Hosts 中的第二个选项，通过设置第二个选项，可以确定过滤的域名以哪种方式来展示。如图 7-33 所示，该设置有 4 个选项，含义分别如下：

- ☐ No Host Filter：不设置域名过滤。
- ☐ Hide the following Hosts：设置的这些与域名相关的会话将在会话列表中隐藏掉不显示。
- ☐ Show only the following Hosts：只在会话列表中显示与设置的这些域名相关的会话。
- ☐ Flag the following Hosts：设置的这些与域名相关的会话将在会话列表中标记显示。

此时，将该选项设置为“Show only the following Hosts”，那么这样，在会话列表中将只显示与在文本框中输入的域名相关的会话。

设置好后，还需要执行会话过滤，此时可以单击“Actions → Run Filterset now”，如图 7-34 所示。

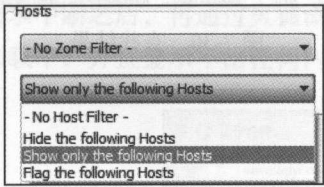


图 7-33 过滤方式的选择

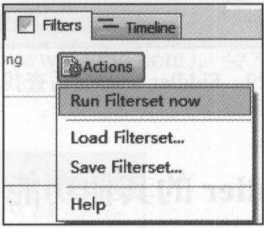


图 7-34 执行会话过滤

执行之后，会发现在会话列表中只留下了与我们设置的域名相关的信息，其他信息则被隐藏掉了，如图 7-35 所示。这样看起来界面会简洁很多，一些我们不关注的网址就不会在此出现了。

630	200	HTTP	blog.csdn.net	/	12.0
679	200	HTTP	www.iqiyi.com	/player/cupid/common/clear.sw...	31.3
702	200	HTTP	www.iqiyi.com	/player/cupid/common/clear.swf?r=69...	31.7
726	200	HTTP	www.iqiyi.com	/player/cupid/common/clear.swf?r=10...	31.7

图 7-35 会话过滤执行结果

## 7.9 小结

1) Fiddler 是一种常见的抓包分析软件,同时,我们可以利用 Fiddler 详细地对 HTTP 请求进行分析,并模拟对应的 HTTP 请求。

2) 网络爬虫是自动爬取网页的程序,在爬取的过程中必然涉及客户端与服务器端之间的通信,自然也需要发送一些 HTTP 请求,并接收服务器返回的结果。在一些稍复杂的网络请求中,我们直接看网址的变化是看不出规律的,此时如果要进行自动化爬取网页,就必须要通过程序构造这些请求,而要通过程序构造这些请求,就必须首先分析这些请求的规律,所以此时我们要使用工具截获这些请求,对这些请求进行分析,这个过程如果使用抓包软件配合进行,则将会变得更加方便。

3) Fiddler 默认监控的地址是 127.0.0.1:8888。

4) Fiddler 是以作为代理服务器的方式进行工作的,所以,本地应用与服务器传递的这些数据都会经过 Fiddler,有的时候,我们希望在传递的中间进行修改后再传递,那么我们可以使用 Fiddler 的断点功能。

打开 Firefox 浏览器以及 Fiddler,然后通过 Firefox 浏览器访问 <http://www.baidu.com>。此时,在 Fiddler 的 Web View 中可以看到百度首页的 HTML 内容。在 Web View 的右侧,可以看到该请求的 Headers 的具体信息,如图 \*2 所示。

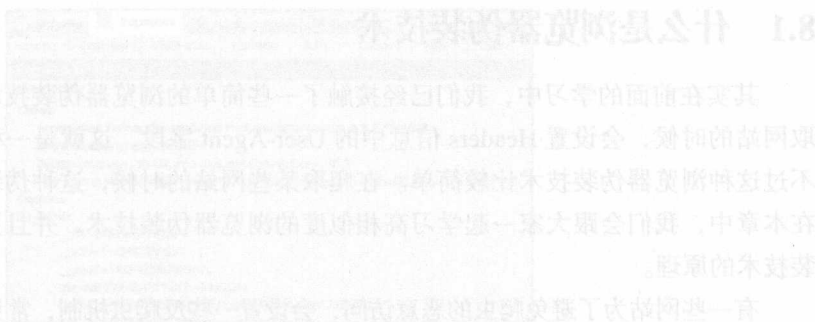
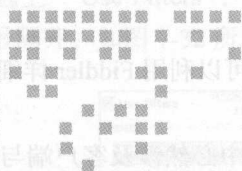


图 7-2 Fiddler 中 Web View 显示请求的 Headers 信息

从图中可以看到,在 Headers 信息中,有很多的字段信息。在了解这些字段信息的含义之前,我们先了解一下 Headers 信息使用的基本原理。

在 HTTP 协议中,客户端与服务器之间通信时,会发送一些请求头 (Request Headers) 和响应头 (Response Headers)。这些头信息包含了关于请求和响应的元数据,如请求的资源、请求的方法、响应的内容类型等。在 Fiddler 中,我们可以查看和修改这些头信息,从而实现对网络流量的分析和控制。





## Chapter 8 第8章

# 爬虫的浏览器伪装技术

有些网站可以识别出访问者是通过浏览器还是爬虫等自动访问程序访问网站，如果识别出使用的不是浏览器，则会禁止访问或者禁止该用户在网站上的其他行为，比如不允许登录等。如果此时我们想对该网站进行爬取，则需要使用浏览器伪装技术。

## 8.1 什么是浏览器伪装技术

其实在前面的学习中，我们已经接触了一些简单的浏览器伪装技术。比如之前我们在爬取网站的时候，会设置 Headers 信息中的 User-Agent 字段，这就是一种浏览器伪装技术，只不过这种浏览器伪装技术比较简单。在爬取某些网站的时候，这种伪装可能并不够用，所以在本章中，我们会跟大家一起学习高相似度的浏览器伪装技术，并且更深入地剖析浏览器伪装技术的原理。

有一些网站为了避免爬虫的恶意访问，会设置一些反爬虫机制，常见的反爬虫机制主要有：

- 1) 通过分析用户请求的 Headers 信息进行反爬虫。
- 2) 通过检测用户行为进行反爬虫，比如通过判断同一个 IP 在短时间内是否频繁访问对应网站等进行分析。
- 3) 通过动态页面增加爬虫爬取的难度，达到反爬虫的目的。

第一种反爬虫机制在目前的网站中应用得最多，这也是我们在本章要解决的问题。一般来说，大部分反爬虫的网站会对用户请求的 Headers 信息的“User-Agent”字段进行检测，以此判断用户的身份，有时，这类反爬虫的网站还会对“Referer”字段进行检测。这些字段的具体含义我们会在下一节中为大家讲解。我们可以在爬虫中构造这些用户请求的 Headers

信息，以此将爬虫伪装成浏览器，简单的伪装只需要设置好“User-Agent”字段的信息即可，如果要进行高相似度的浏览器伪装，则需要将用户请求的 Headers 信息中常见的字段都在爬虫中设置好。

第二种反爬虫机制的网站，可以通过之前学习的使用代理服务器并经常切换代理服务器的方式，一般就能够攻克限制。

第三种反爬虫机制的网站，可以利用一些工具软件，比如 selenium+phantomJS，就可以攻克限制。

在此，我们将主要为大家分析如何攻克反爬虫网站的第一类限制机制，这里会采用高相似度的浏览器伪装技术突破该类限制。

## 8.2 浏览器伪装技术准备工作

在学习高相似度的浏览器伪装技术之前，我们首先要对 Headers 信息的基本原理与结构进行相应了解。

打开火狐浏览器以及 Fiddler，然后通过火狐浏览器访问优酷官网，随后可以在 Fiddler 中监控到对应的会话信息，如图 8-1 所示。

单击该会话信息，在该会话信息的右方的界面中，将标签切换为“Inspectors → Headers”，然后可以看到用户请求的 Headers 的具体信息，如图 8-2 所示：

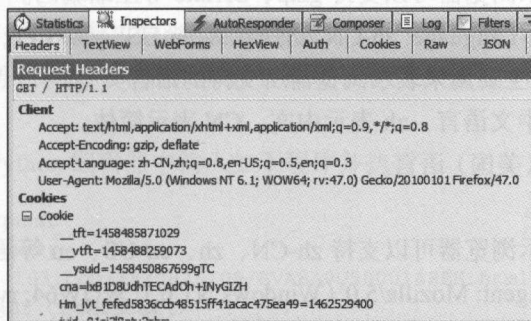


图 8-2 Fiddler 中监控的会话信息中的 Headers 信息

可以发现，在这里有很多的字段信息，在了解这些字段信息的含义之前，我们先简单了解一下 Headers 信息使用的基本原理。

当我们通过浏览器访问某个网址的时候，会向服务器发送一些 Headers 头信息，然后服务器会根据对应的用户请求头信息生成一个网页内容，并将生成的内容返回给浏览器。这就是 Headers 头信息使用的基本原理和过程，在这个过程中，当服务器接收到这些头信息之后，可以知道当前浏览器的状态，同样服务器也可以根据当前浏览器的状态分析对应请求的用户

是爬虫的可能性有多大,从而决定是否做出响应以及做出什么样的响应。

要使用这些头信息,首先需要知道这些头信息中常见的各字段是什么含义。

首先我们需要知道各字段的格式,基本格式为:“字段名:字段值”,比如字段信息“Accept-Language: zh-CN, zh; q=0.8, en-US; q=0.5, en; q=0.3”中“Accept-Language”为字段名,“zh-CN, zh; q=0.8, en-US; q=0.5, en; q=0.3”为该字段名对应的值。字段名与对应的值之间通过“:”隔开。

接下来,我们跟大家一起学习一下常见头信息中字段的含义。

常见字段 1: Accept: text/html, application/xhtml+xml, application/xml; q=0.9, \*/\*; q=0.8

□ Accept 字段主要用来表示浏览器能够支持的内容类型有哪些。

□ text/html 表示 HTML 文档。

□ application/xhtml+xml 表示 XHTML 文档。

□ application/xml 表示 XML 文档。

□ q 代表权重系数,值介于 0 和 1 之间。

所以这一行字段信息表示浏览器可以支持 text/html、application/xhtml+xml、application/xml、\*/\* 等内容类型,支持的优先顺序从左到右依次排列。

常见字段 2: Accept-Encoding: gzip, deflate

□ Accept-Encoding 字段主要用来表示浏览器支持的压缩编码有哪些。

□ gzip 是压缩编码的一种。

□ deflate 是一种无损数据压缩算法。

这一行字段信息表示浏览器可以支持 gzip、deflate 等压缩编码。

常见字段 3: Accept-Language: zh-CN, zh; q=0.8, en-US; q=0.5, en; q=0.3

□ Accept-Language 主要用来表示浏览器所支持的语言类型。

□ zh-CN 表示简体中文语言, zh 表示中文, CN 表示简体。

□ en-US 表示英语(美国)语言。

□ en 表示英语语言。

所以之一行字段表示浏览器可以支持 zh-CN、zh、en-US、en 等语言。

常见字段 4: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0

□ User-Agent 字段主要表示用户代理,服务器可以通过该字段识别出客户端的浏览器类型、浏览器版本号、客户端的操作系统及版本号,网页排版引擎等客户端信息。所以我们之前要模拟浏览器登录,主要以伪造该字段进行。

□ Mozilla/5.0 表示浏览器名及版本信息。

□ Windows NT 6.1; WOW64; rv:47.0 表示客户端操作系统对应信息。

□ Gecko 表示网页排版引擎对应信息。

□ firefox 自然表示火狐浏览器。

所以这一行字段表示的信息为对应的用户代理信息是 Mozilla/5.0 (Windows NT 6.1; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0。

常见字段 5: Connection: keep-alive

□ Connection 表示客户端与服务器的连接类型，对应的字段值主要有两种：

- keep-alive 表示持久性连接。
- close 表示单方面关闭连接，让连接断开。

所以此时，这一行字段表示客户端与服务器的连接是持久性连接。

常见字段 6: Host: www.youku.com

□ Host 字段表示请求的服务器网址是什么，此时这一行字段表示请求的服务器网址是 www.youku.com。

常见字段 7: Referer: 网址

□ Referer 字段主要表示来源网址地址，比如我们从 http://www.youku.com 网址中访问了该网址下的子页面 http://tv.youku.com/?spm=0.0.topNav.5 ~ 1 ~ 3!2 ~ A.QnQOEf, 那么此时来源网址为 http://www.youku.com，即此时 Referer 字段的值为 http://www.youku.com。

通过上面的学习，我们已经知道了 Headers 信息中常见的各字段的含义，清楚了这些含义之后，我们就能够根据自身的需求构造出对应 Headers 的数据了。

## 8.3 爬虫的浏览器伪装技术实战

接下来我们通过实例讲解爬虫的浏览器伪装技术。

首先我们通过实例 1 来看一下若没有设置浏览器伪装，通过 Fiddler 监控到的是什么会话信息。

打开 Fiddler，在 Python 编辑器中输入如下程序并执行：

```
import urllib.request
import http.cookiejar
url= "http://news.163.com/16/0825/09/BVA8A9U500014SEH.html"
cjar=http.cookiejar.CookieJar()
proxy= urllib.request.ProxyHandler({'http':'127.0.0.1:8888'})
opener = urllib.request.build_opener(proxy, urllib.request.HTTPHandler,urllib.
request.HTTPCookieProcessor(cjar))
urllib.request.install_opener(opener)
data=urllib.request.urlopen(url).read()
fhandle=open("D:/Python35/myweb/part8/1.html","wb")
fhandle.write(data)
fhandle.close()
```

程序中使用 127.0.0.1:8888 作为代理服务器，这样就可以通过 Fiddler 截获对应的会话信息，随后程序爬取了网易新闻中的一个网页，并保存到本地。



执行后,我们发现,此时在 Fiddler 中截获了对应的会话信息,如图 8-3 所示:

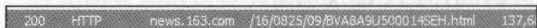


图 8-3 Fiddler 中监控到的爬虫会话信息

单击该会话信息,在右方窗口中切换到“Inspectors → Headers”,可以看到该会话的具体用户请求 Headers 信息,如图 8-4 所示。可以看到,此时的用户代理为“Python-urllib/3.5”,允许的压缩编码类型为 identity,服务器在收到这些 Headers 信息之后,就可以对客户端的基本情况进行分析。

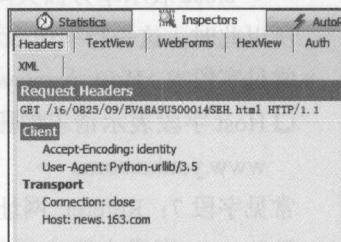


图 8-4 Fiddler 中监控到的爬虫会话信息中的 Headers 值

打开保存在本地的文件,如图 8-5 所示,说明此时已经成功爬取了对应的网页。由此我们可以知道,该网站没有建立这一项反爬虫机制,但是目前很多网站都会设置这一项反爬虫机制,所以比较保险的做法是,不管这些网站是否具有反爬虫机制,都伪装成浏览器去爬取,而伪装成浏览器去爬取不同网站的代码基本上是一致的,所以实现起来也很方便。

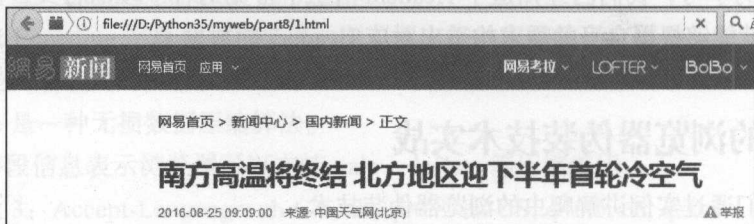


图 8-5 爬虫爬到的网页

那么,应当如何伪装成浏览器呢?

首先,我们需要设置好对应用户请求的 Headers 信息,在 Python 中,我们可以通过 opener.addheaders 为爬虫添加 Headers 信息,但是此时添加的 Headers 信息要具备指定的格式,格式为: [(字段名 1, 对应的值 1), (字段名 2, 对应的值 2), ..., (字段名 n, 对应的值 n)]。

所以,我们需要将对应的 Headers 信息设置为如下格式:

```
[('Connection', 'keep-alive'), ('referer', 'http://www.163.com/'), ('Accept-Language', 'zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3'), ('User-Agent', 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0'), ('Accept', 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8')]
```

可以看到,对应的格式外层为一个列表,里层为元组,各元组间通过逗号隔开。

此时,我们可以先将 Headers 中各字段信息先以字典的形式赋值给一个变量,然后再构建一个新变量,该新变量中存储一个空列表,随后,通过 for 循环遍历上述的字典类型的变

量并重构为元组，然后在每次 for 循环中将构建的新的元组添加到列表类型的变量中，作为列表类型变量的新元素。

所以，我们可以改进一下，即可得到实例 2：

```
import urllib.request
import http.cookiejar
url= "http://news.163.com/16/0825/09/BVA8A9U500014SEH.html"
# 以字典的形式设置 headers
headers={ "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
          "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
          "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0",
          "Connection": "keep-alive",
          "referer": "http://www.163.com/" }

# 设置 cookie
cjar=http.cookiejar.CookieJar()
proxy= urllib.request.ProxyHandler({'http': "127.0.0.1:8888"})
opener = urllib.request.build_opener(proxy, urllib.request.HTTPHandler,urllib.request.HTTPCookieProcessor(cjar))
# 建立空列表，为了以指定格式存储头信息
headall=[]
# 通过 for 循环遍历字典，构造出指定格式的 headers 信息
for key,value in headers.items():
    item=(key,value)
    headall.append(item)
# 将指定格式的 headers 信息添加好
opener.addheaders = headall
# 将 opener 安装为全局
urllib.request.install_opener(opener)
data=urllib.request.urlopen(url).read()
fhandle=open("D:/Python35/myweb/part8/2.html", "wb")
fhandle.write(data)
fhandle.close()
```

程序中，为了方便调试，同样将代理服务器设置为 127.0.0.1:8888，随后设置好 headers 信息，并将构建的指定格式的 headers 信息通过 opener.addheaders 添加到爬虫中。随后读取对应网页并写入到指定的本地文件中。

代码执行后，我们可以看到 Fiddler 中截获了图 8-6 所示的会话信息。




图 8-6 Fiddler 中监控到的爬虫会话信息

单击该会话信息，在右边的窗口中切换到“Inspectors → Headers”，可以看到如图 8-7 所示的信息。可以看到，此时所显示的用户请求的 Headers 信息已经跟通过浏览器访问对于网址时的用户请求的 Headers 信息基本上一致了，说明此时爬虫已经伪装成浏览器了。

同样，使用该程序我们也成功爬取了对应网页，打开“D:/Python35/myweb/part8/2.html”，可以看到爬取的结果如图 8-8 所示：

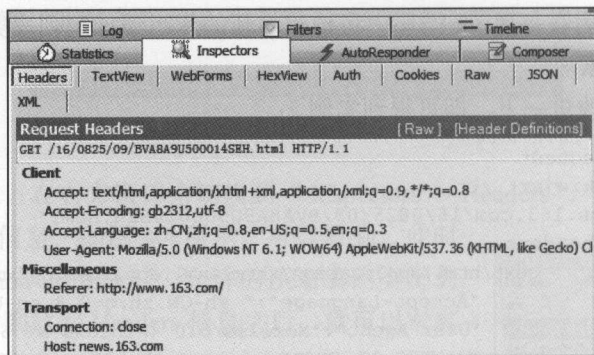


图 8-7 Fiddler 中监控到的爬虫会话信息的 Headers 值

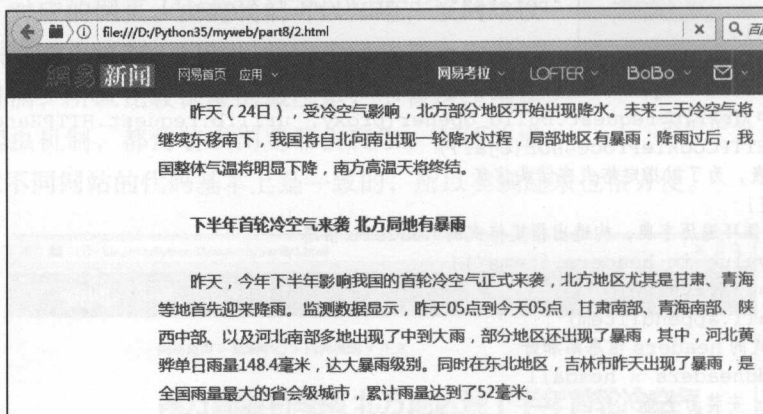


图 8-8 爬虫爬取到的网页

值得注意的是：

1) 如果 Accept-Encoding 设置为 gzip, deflate, 有可能会出现乱码问题, 此时只需要将该字段信息省略不写或者将该字段信息的值设置为 utf-8 或 gb2312 即可。为什么将该字段的值设置为 gzip, deflate 会出问题呢? 是因为如果设置该字段为 gzip, deflate, 那么从服务器返回来的是对应的 gzip, deflate 压缩的代码, 此时没有进行解码, 故而会出现乱码的情况, 而一些常规浏览器中, 从服务器返回对应的 gzip, deflate 压缩的代码后, 浏览器可以自动进行解压缩, 故而不会出现乱码。

2) 使用 Fiddler 作为代理服务器, 所爬取的网址要以具体文件或者 “/” 结尾, 如果有具体文件, 直接写该具体文件的网址即可, 比如将要爬取的网址写为: “http://news.163.com/16/0825/09/BVA8A9U500014SEH.html” 这种写法是合法的, 如果被爬取网址是一个文件夹, 比如要爬取 “http://www.baidu.com”, 此时爬取的是一个目录 (文件夹), 所以需要以 “/” 结尾, 如 “http://www.baidu.com/” 这种写法是合法的, 而 “http://www.baidu.com” 这种写法则可能会出现相应错误。

3) referer 字段的值一般可以设置为要爬取的网页的域名地址或对应网站的主页网址。

在实际项目中,我们不一定要将 Fiddler 设置为代理服务器来伪装成浏览器,在此设置 Fiddler 为代理服务器主要为了方便抓包,从而方便调试。所以,在实际项目中,我们可以将该项过程省略,可以对上面的代码进行改进,改进后,程序如下所示,此时,由于不需要进行相应调试,所以我们已经去掉了代理服务器的设置过程:

```
import urllib.request
import http.cookiejar
# 注意,如果要通过 fiddler 调试,则下方网址要设置为 "http://www.baidu.com/"
url= "http://www.baidu.com"
headers={ "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
          "Accept-Encoding": "gb2312,utf-8",
          "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
          "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0",
          "Connection": "keep-alive",
          "referer": "baidu.com"}

cjar=http.cookiejar.CookieJar()
opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cjar))
headall=[]
for key,value in headers.items():
    item=(key,value)
    headall.append(item)
opener.addheaders = headall
urllib.request.install_opener(opener)
data=urllib.request.urlopen(url).read()
fhandle=open("D:/Python35/myweb/part8/3.html","wb")
fhandle.write(data)
fhandle.close()
```

在实际项目中,我们可以使用上方代码,稍微改变一下,即可以将爬虫高相似度地模拟成浏览器。

## 8.4 小结

1) 常见的反爬虫机制主要有:通过分析用户请求的 Headers 信息进行反爬虫;通过检测用户行为进行反爬虫,比如通过判断同一个 IP 在短时间内是否频繁访问对应网站等进行分析;通过动态页面增加爬虫爬取的难度,达到反爬虫的目的。

2) 使用 Fiddler 作为代理服务器,所爬取的网址要以具体文件或者 "/" 结尾。

3) referer 字段的值一般可以设置为要爬取的网页的域名地址或对应网站的主页网址。

4) 在实际项目中,要伪装成浏览器,我们不一定要将 Fiddler 设置为代理服务器,上面实例 1、2 设置 Fiddler 为代理服务器主要为了方便抓包,从而方便调试。所以,在实际项目中,我们可以将该项过程省略。



## 爬虫的定向爬取技术

在本书第1章的爬虫理论知识中，我们提到过，爬虫有多种类型，其中比较典型的两种类型就是通用网络爬虫与聚焦网络爬虫。在聚焦网络爬虫中，我们常常需要根据设置的主题定向爬取，此时需要用到爬虫的定向爬取技术。

### 9.1 什么是爬虫的定向爬取技术

谈到爬虫的定向爬取技术，其实我们并不陌生，在前几章内容中，我们已经初步见到了爬虫的定向爬取技术。在本章中，将会系统地学习爬虫的定向爬取技术，并会为大家实现一个相对来说有些难度的采用定向爬取技术的爬虫。

通俗来说，爬虫的定向爬取技术就是根据设置的主题，对要爬取的网址或者网页中的内容进行筛选。比如我们可以使用正则表达式进行筛选等，筛选之后，再爬取对应的网址中的内容，并可以根据爬取到的内容再次进行筛选。

众所周知，互联网的信息是海量的，在一个相对较短的时间内要尽可能多的爬取到我们感兴趣的信息，则不可能漫无目的地去爬取，如果漫无目的地去爬取，则必然会浪费大量的时间，所以我们需要根据我们设置的主题，拟定出对应的爬取策略与爬取规则，这样，才可以让我们在较短的时间内从海量的互联网信息中尽可能多的爬取出与主题相关的信息。而根据设定的主题建立爬虫的爬取策略与爬取规则是爬虫的定向爬取技术的核心与重点部分。

具体来说，爬虫的定向爬取技术主要需要解决3个问题：

- 1) 清晰地定义好爬虫的爬取目标，规划好主题。
- 2) 建立好爬取网址的过滤筛选规则以及内容的过滤筛选规则。

3) 建立好 URL 排序算法, 让爬虫能够明确优先爬取哪些网页、以什么顺序爬取待爬取的网页。比如, 待爬取的 URL 网址可能有很多, 在爬虫资源有限的情况下, 需要确定好这些网址的爬行顺序, 以不一样的顺序去爬取, 可能会导致不一样的爬取效率。

## 9.2 定向爬取的相关步骤与策略

关于聚焦网络爬虫定向爬取网页的相关算法以及爬行策略我们在本书的前三章爬虫的理论知识部分已进行了详细阐述, 在此不再赘述, 这里我们主要为大家讲解爬虫定向爬取的相关步骤以及在 Python 爬虫中, 实际进行信息筛选的方法和策略主要有哪些。

在一个爬虫项目中, 定向爬取某些信息的步骤主要有:

1) 理清爬取的目的。这一步非常关键, 有一个明确的爬取目的, 可以让我们在设置爬取规则的时候思路更加清晰, 爬取失败率更低。

2) 设置网址的过滤规则。这一步显然不是必须的, 但是在网址数较多的爬取任务中, 合理地进行该项设置, 可以大大提高爬虫的爬取效率。由于有的时候爬虫爬取的网址数量很多, 要爬取的内容在某些有规律的网址中, 此时, 我们可以设置对应的模式, 比如设置好对应的正则表达式, 将不满足格式的网址过滤掉, 此时爬虫就不需要爬取那些没有包含目标信息的网址了, 爬虫只需要爬取满足格式的网址, 即包含目标信息的网址即可, 这样做可以大大的提高爬行效率, 当然这一步并不是强制要求去做的。对于某些网址数量并不多的爬取任务, 我们是否进行该项设置对爬取效率的影响并不会太大。

3) 设置好内容采集规则。通过这一步的设置, 我们可以提取出我们关注的信息, 从而过滤掉那些不关注的信息, 信息筛选的方法与策略有很多, 通过正则表达式去筛选信息是其中一种方法。

4) 规划好采集任务, 合理的设置爬虫线程与爬虫数量。对于任务量不大的爬虫, 使用一个单线程爬虫即可完成。但如果爬虫的任务量很大, 此时为了提高效率, 我们可以使用多线程爬虫或者使用多个爬虫去爬取对应的任务, 但是, 如果使用多线程爬虫或使用多个爬虫去爬取, 则需要对每个爬虫要爬取的任务进行合理规划, 避免出现一直重复爬取或某些目标网页漏爬的情况。

5) 将采集结果进行相应的修正, 处理成我们想要的格式。完成采集后, 有可能采集结果并不是我们想要的格式, 此时我们对采集的结果进行相应的修正, 比如进行编码、解码、格式整理等修正操作, 将采集的结果处理成我们需要的格式。

6) 对结果进行进一步处理, 完成任务。比如, 如有需要, 我们可以将结果写入数据库等, 等相应的后续处理操作, 从而完成我们的爬虫任务。

这些过程, 可以通过图 9-1 形象地表示出来。

在上面的过程中, 信息筛选是一个重点的核心步骤, 无论是在网址过滤还是在内容采集中都会用到。在 Python 中, 进行信息筛选的方法策略主要有:

1) 通过正则表达式筛选。这也是我们之前常用的一种信息筛选的方法,通过正则表达式我们可以构造出对应的信息模式,从而可以根据这个模式匹配出符合格式的内容。

2) 通过 XPath 表达式筛选。XPath 是 XML 的路径语言,我们可以利用该语言快速地在 XML 文档中查找出对应的信息,当然也可以快速地对网页进行定位,XPath 表达式是使用 XPath 语言所写的表达式,我们可以使用不同的 XPath 表达式筛选出不同的信息。我们在后面的 Scrapy 爬虫框架的学习中,经常会使用到 XPath。

3) 通过 xslt 筛选。除了上面两种方法之外,有时很多朋友也会用 xslt 来提取网页的数据。

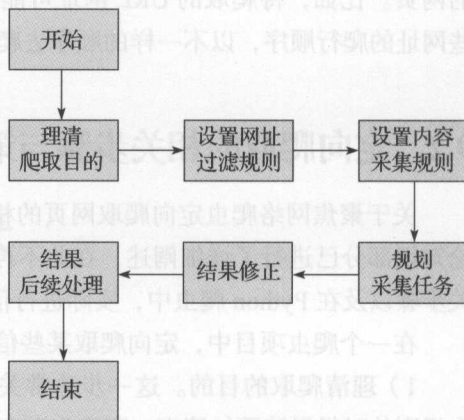


图 9-1 定向爬取的主要步骤

当然,除了上面这 3 种信息筛选的方法策略

之外,还有很多其他的信息筛选方式,由于用得较少,故而未提。

通过上面的学习,我们已经基本上知道了在一个爬虫项目中,定向爬取某些信息的步骤主要有哪些,以及在 Python 中对信息的筛选方法主要有哪些,了解这些基础知识之后,我们在下一节会为大家讲解一个定向爬取的实例。

## 9.3 定向爬取实战

假如,我们若要对腾讯视频中的某个视频的评论进行批量爬取并实现自动加载新评论,就需要用到爬虫的定向爬取技术。与以往实例不同的是,腾讯视频中的评论,若要显示更多新评论,不会在 URL 网址中变化成下一页,而是在当前页面中进行加载,所以我们会通过 Fiddler 对网页行为进行分析,并实现评论的自动加载。

假如要爬取评论的视频网址是: <http://v.qq.com/x/cover/0lmgk2kez0lztri.html>, 首先,我们需要开启 Fiddler,通过火狐浏览器手动打开该网址并进行相应分析,我们打开的视频是最近的一部电视连续剧《九州天空城》,如图 9-2 所示。



图 9-2 《九州天空城》视频页面

随后,将网页拖动到评论处,单击“加载更多”,如图 9-3 所示。

接着,切换到 Fiddler 的会话列表页面,发现捕获到了如图 9-4 所示的会话信息,这个会

话信息就是加载评论时所触发的真实网址。

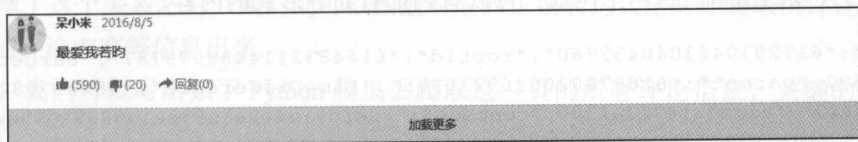


图 9-3 视频页面中的对应评论信息，可以单击“加载更多”

Result	Protocol	Host	URL
200	HTTP	coral.qq.com	/article/1472528692/comment?commen...

图 9-4 Fiddler 截获的会话信息

单击该会话信息，可以看到该会话信息的头部请求详情，如图 9-5 所示。

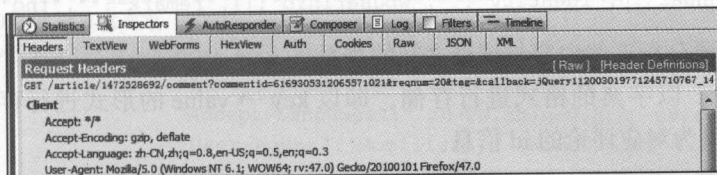


图 9-5 Fiddler 截获的会话信息详情

将触发的网址复制出来，此次的网址为：

```
http://coral.qq.com/article/1472528692/comment?commentid=6169305312065571021&reqnum=20&tag=&callback=jQuery112003019771245710767_1472204567039&_=1472204567042
```

为了便于观察加载评论信息时触发的网址之间的规律，再次在视频网页中的评论处单击“加载更多”，用同样的方法可以在 Fiddler 中截获到新触发的对应的真实网址，此次的网址为：

```
http://coral.qq.com/article/1472528692/comment?commentid=6173403130078248384&reqnum=20&tag=&callback=jQuery1120033349277085574114_1472205336515&_=1472205336517
```

观察这两个网址之间的差异，可以分析出如下结果：

- 1) 两次 commentid 字段的值不一样，该字段代表的是评论的 id。
- 2) 两次 reqnum 字段的值都是 20，观察每次加载的评论数，发现也是 20 条，所以可以推断得出，reqnum 字段代表的是每次评论加载的数量。
- 3) 两次都出现了 1472528692，可以分析并推断得出这个值为对应视频的一种编号，即 1472528692 代表的是《九州天空城》的视频评论。
- 4) 网址中后续的“&tag=&callback=jQuery1120033349277085574114\_1472205336515&\_=1472205336517”等字段信息可以省略，我们可以省略后再访问该网址进行验证。
- 5) 所以可以得到，视频评论的 URL 地址格式为“http://coral.qq.com/article/ 视频编号 /



comment?commentid= 评论编号 &reqnum=20"。

我们打开复制出来的这两个网址, 可以看到都有如下形式的内容。

```
{
  "id": "6172930423040432760",
  "rootid": "6164832114488979787",
  "targetid": 1472528692,
  "parent": "6168760205369235005",
  "timeDifference": "08\u670821\u65e509:01:40",
  "time": 1471741300,
  "content": "\u8fd9\u4e2a\u95ee\u9898\u95ee\u5f97\u597d",
  "title": "",
  "up": 123,
  "rep": 1,
  "type": 1,
  "hotscale": 0,
  "checktype": 2,
  "checkstatus": 1,
  "isdeleted": 0,
  "tagself": "",
  "taghost": "",
  "source": 9,
  "location": "",
  "address": "",
  "rank": -1,
  "custom": "",
  "extend": {
    "at": 0,
    "ut": 0
  },
  "orireplynum": 0,
  "richtype": 0,
  "userid": "76607389",
  "poke": 6,
  "abstract": "",
  "thiridid": "",
  "replyuser": "\u6613\u5982\u65e2\u5f80\u265b\u4e3a\u674e\u5cf0\u72c2\u5f5e",
  "replyuserid": "142837631",
  "replyhwvip": 1,
  "replyhwlevel": 1,
  "replyhwannual": 0,
  "userinfo": {
    "userid": "76607389",
    "uidex": "ec8c09ba112afbc2fd7ac7602b54480d92",
    "nick": "\u5fc3\u5982\u8584\u8377\u5929\u7136\u51c9",
    "head": "http://q4.qlogo.cn/g?b=qq&k=icIL8y4ZgXcHlhcjPf7zIJQ&s=40&t=1472140800",
    "gender": 2,
    "viptype": 0,
    "mediaid": 0,
    "region": "\u68b5\u8482\u5188",
    "thirdlogin": 0,
    "hwvip": 0,
    "hwlevel": 0,
    "hwannual": 0,
    "identity": "",
    "wbuserinfo": [],
    "remark": "",
    "fnd": 0
  }
}
```

接下来我们要分析这些内容与评论之间的关系。

这些内容中, 以字典的格式进行存储, 即以 key → value 的形式进行存储。有一项为 "id" 字段, 这一项为对应评论的 id 信息。

我们继续观察, 发现有一些需要进行 unicode 编码之后才能够显示的内容, 主要有:

- 1) "content": "\u8fd9\u4e2a\u95ee\u9898\u95ee\u5f97\u597d"
- 2) "replyuser": "\u6613\u5982\u65e2\u5f80\u265b\u4e3a\u674e\u5cf0\u72c2\u5f5e"
- 3) "nick": "\u5fc3\u5982\u8584\u8377\u5929\u7136\u51c9"

我们可以分别对这 3 项字段信息中的 value 部分进行 unicode 编码, 并输出对应的结果, 如图 9-6 所示。

此时, 我们就可以很清晰地得到了这 3 个字段名与字段值的对应关系:

- 1) "content" → '这个问题问得好'
- 2) "replyuser" → '易如既往 ㊗ 为李峰狂 ~'
- 3) "nick" → '心如薄荷天然凉'

随后, 我们打开该视频的网页, 找到对应的评论信息, 如图 9-7 所示:

```
>>> u"\u8fd9\u4e2a\u95ee\u9898\u95ee\u5f97\u597d"
'这个问题问得好'
>>> u"\u6613\u5982\u65e2\u5f80\u265b\u4e3a\u674e\u5cf0\u72c2\u5f5e"
'易如既往 ㊗ 为李峰狂 ~'
>>> u"\u5fc3\u5982\u8584\u8377\u5929\u7136\u51c9"
'心如薄荷天然凉'
```

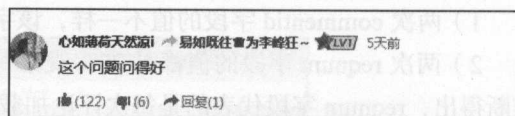


图 9-6 通过 Python 编码对应信息得到的结果

图 9-7 视频页面中对应的评论信息

通过对比, 我们可以很容易发现这些字段名与评论的关系如下:

- 1) "content" → 具体评论内容
- 2) "replyuser" → 该评论是回复谁的

### 3) "nick"→评论用户的昵称

理清了这个关系之后,我们则可以构造对应的正则表达式定向地爬取出评论的用户名以及具体的评论内容等信息出来。

所以,我们可以写出如下 Python 爬虫去爬取这一页的所有评论信息,关键的地方都给出了注释。

```
import urllib.request
import http.cookiejar
import re
# 设置视频编号
vid="1472528692"
# 设置评论起始编号
comid="6173403130078248384"
# 构造出真实评论请求网址
url= "http://coral.qq.com/article/"+vid+"/comment?commentid="+comid+"&reqnum=20"
# 设置头信息伪装成浏览器爬取
headers={ "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
           "Accept-Encoding": "gb2312,utf-8",
           "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
           "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0", "Connection": "keep-alive",
           "referer": "qq.com"}

# 设置 cookie
cjar=http.cookiejar.CookieJar()
opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cjar))
headall=[]
for key,value in headers.items():
    item=(key,value)
    headall.append(item)
opener.addheaders = headall
urllib.request.install_opener(opener)
# 爬取该网页
data=urllib.request.urlopen(url).read().decode("utf-8")
# 分别构建筛选 id、用户名、评论内容等信息的正则表达式
idpat='id":"(.*?)"'
userpat='nick":"(.*?)",'
conpat='content":"(.*?)",'
# 分别根据正则表达式查找所有 id、用户名、评论内容等信息
idlist=re.compile(idpat,re.S).findall(data)
userlist=re.compile(userpat,re.S).findall(data)
conlist=re.compile(conpat,re.S).findall(data)
# 通过循环将获取到的各信息遍历出来
for i in range(0,20):
    # 输出对应信息,并对字符串进行 unicode 编码,从而正常显示
    print(" 用户名是 :"+eval('u'+userlist[i]+''))
    print(" 评论内容是 :"+eval('u'+conlist[i]+''))
    print("\n")
```

该程序会伪装成浏览器对相关评论页面进行爬取，爬取后会通过正则表达式将对应信息提取出来，然后通过 for 循环将每条评论信息分别输出。

执行结果如下所示，由于篇幅所限，获取的评论内容较多，在“……”处省略了部分执行结果：

```
>>>
===== RESTART: D:\Python35\9.3.py =====
用户名是：心如薄荷天然凉 i
评论内容是：这个问题问得好

用户名是：`颖~
评论内容是：没有你看怒火 v 很符合国家

用户名是：X秒殺于無形
评论内容是：这，怎么

用户名是：小二，来碗泪流满面
评论内容是：剧本里就是下雪的呀
.....
```

这个程序有一个缺点，就是无法加载新的评论。那么应当怎样实现加载新评论的功能呢？

我们知道，真实评论请求网址中的 commentid 字段代表的是起始评论的 ID。所以此时，我们只需要将每次评论页面中的最后一个评论 ID 提取出来，并赋值到 URL 网址中的对应位置，即可构造出新的 URL 网址，从而加载出新的评论。

所以，我们可以对上述代码进行相应改进，即可实现自动加载新评论的功能。

改进后的参考代码如下，代码中关键位置给出了注释。

```
import urllib.request
import http.cookiejar
import re
# 视频编号
vid="1472528692"
# 刚开始时候的评论 ID
comid="6173403130078248384"
url= "http://coral.qq.com/article/"+vid+"/comment?commentid="+comid+"&reqnum=20"
headers={ "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
           "Accept-Encoding": "gb2312,utf-8",
           "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
           "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0", "Connection": "keep-alive",
           "referer": "qq.com" }
```

```

cjar=http.cookiejar.CookieJar()
opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cjar))
headall=[]
for key,value in headers.items():
    item=(key,value)
    headall.append(item)
opener.addheaders = headall
urllib.request.install_opener(opener)
# 建立一个自定义函数 craw(vid,comid), 实现自动爬取对应评论网页并返回爬取数据
def craw(vid,comid):
    url= "http://coral.qq.com/article/"+vid+"/comment?commentid="+comid+"&regnum=20"
    data=urllib.request.urlopen(url).read().decode("utf-8")
    return data
idpat='"id": "(.*)"'
userpat='"nick": "(.*)"'
conpat='"content": "(.*)"'
# 第一层循环, 代表爬取多少页, 每一次外层循环爬取一页
for i in range(1,10):
    print("-----")
    print("第 "+str(i)+" 页评论内容 ")
    data=craw(vid,comid)
    # 第二层循环, 根据爬取的结果提取并处理每条评论的信息, 一页 20 条评论
    for j in range(0,20):
        idlist=re.compile(idpat,re.S).findall(data)
        userlist=re.compile(userpat,re.S).findall(data)
        conlist=re.compile(conpat,re.S).findall(data)
        print(" 用户名是 :"+eval('u'+userlist[j]+''))
        print(" 评论内容是 :"+eval('u'+conlist[j]+''))
        print("\n")
    # 将 comid 改变为该页的最后一条评论 id, 实现不断自动加载
    comid=idlist[19]

```

该程序同样会伪装成浏览器对对应评论页面进行爬取, 此时, 我们将爬取的过程封装成了一个函数, 这样就可以根据不同的参数传递, 从而实现爬取不同的评论网页。同时, 进行了两层 for 循环, 第一层 for 循环实现爬行页数的控制, 每次循环爬取一页, 第二层 for 循环实现对每页的评论进行分别输出处理, 每次循环输出一条评论信息。完成第二层 for 循环后, 需要在第一层 for 循环的末尾处对 comid 进行重新赋值, 从而实现新评论的加载。

程序的执行结果如下:

```
>>>
```

```
===== RESTART: D:\Python35\9.3.py =====
```

```
第 1 页评论内容
```

```
用户名是: 心如薄荷天然凉 i
```

```
评论内容是: 这个问题问得好
```

```
用户名是: `颖~
```



评论内容是：没有你看怒火 v 很符合国家

用户名是：X秒殺于無形

评论内容是：这，怎么

用户名是：小二，来碗泪流满面

评论内容是：剧本里就是下雪的呀

.....

第 9 页评论内容

用户名是：----

评论内容是：明明说

用户名是：后来、

评论内容是：这几个男演员真是丑到家

用户名是：捷

评论内容是：对

.....

同样，由于篇幅有限，“.....”处省略了部分执行结果。可以看到，此时已经成功爬取了 9 页的评论信息，并且实现了自动加载新内容的功能。

通过这个案例的学习，相信大家对爬虫的定向爬取技术已经有了比较深入的了解，并且懂得了如何结合 Fiddler 来分析网页的会话信息，学会了如何在同一个页面中自动加载新内容的方法，希望可以抛砖引玉，让大家写出更多优质的网络爬虫。

## 9.4 小结

1) 通俗来说，爬虫的定向爬取技术就是根据设置的主题，对要爬取的网址或者网页中的内容进行筛选，比如我们可以使用正则表达式进行筛选等，筛选之后，再爬取对应的网址中的内容，并可以根据爬取到的内容再次进行筛选。

2) 互联网的信息是海量的，我们在一个相对较短的时间内要尽可能多的爬取到我们感兴趣的信息，则不可能漫无目的地去爬取，如果漫无目的地去爬取，则必然会浪费大量的时间，所以我们需要根据设置的主题，拟定出对应的爬取策略与爬取规则，这样，才可以让我们在较短的时间内从海量的互联网信息中尽可能多的爬取出与主题相关的信息。而根据设定的主题建立爬虫的爬取策略与爬取规则是爬虫的定向爬取技术的核心与重点部分。

3) 在 Python 中，进行信息筛选的方法策略主要有：通过正则表达式筛选、通过 XPath 表达式筛选、通过 xslt 筛选。

### 第三篇 *Part 3*

## 框架实现篇

在这一章中，我们会带大家了解 Python 的爬虫框架，包括什么爬虫框架以及 Python 中常见的爬虫框架，让大家对 Python 爬虫框架有一个基本的认识。

### 10.1 什么是 Python 爬虫框架

简单来说，Python 的爬虫框架就是一些爬虫项目的半成品。我们可以将一些常见爬虫功能的实现代码部分写好，然后留下一些接口，在做不同的爬虫项目时，我们只需要根据实际情况，编写步骤代码，然后调用这些接口，即可以实现一个爬虫项目。

- 第 10 章 了解 Python 爬虫框架
- 第 11 章 爬虫利器——Scrapy 安装与配置
- 第 12 章 开启 Scrapy 爬虫项目之旅
- 第 13 章 Scrapy 核心架构
- 第 14 章 Scrapy 中文输出与存储
- 第 15 章 编写自动爬取网页的爬虫
- 第 16 章 CrawlSpider
- 第 17 章 Scrapy 高级应用

### 10.2 常见的 Python 爬虫框架

在 Python 中，开源爬虫框架有诸多，甚至我们也可以自己写一些爬虫框架。我们并不

评论内容是：这篇文章对爬虫很有帮助

用户名是：王梦溪于老师

前面章节中，我们的爬虫项目都是一步步写出来的，使用纯手写的方式来实现这些项目，相对来说会慢一些，如果我们有一套开发相对完备的框架，那么写少量代码就可以实现一样甚至更复杂的功能。因为很多代码都已经事先在框架中封装好了，从而大大提高了项目开发的效率。在接下来的学习中，我们会重点介绍框架爬虫项目的开发。

评论内容是：完美。

评论内容是：这个内容真是受益匪浅

用户名是：我

评论内容是：好

同样，由于篇幅有限，“……”处省略了部分执行结果。可以看到，此时已经成功爬取了9页的评论信息，并且实现了自动加载新内容的功能。

通过这个案例的学习，相信大家对爬虫的定向爬取技术已经有了比较深入的了解，并且懂得如何将结合 BeautifulSoup 来解析网页的会话信息，学会了如何在同一个页面中自动加载新内容的方法，希望可以抛砖引玉，让大家写出更多优质的网络爬虫。

## 9.4 小结

1) 通俗来说，爬虫的定向爬取技术就是根据指定的主题，对要爬取的网页或者网页中的内容进行筛选，比如我们可以使用正则表达式来提取网页中的内容，并可以根据爬取到的内容来进一步筛选内容。

2) 互联网的信息是海量的，我们可以在一个指定的时间段内，对指定的主题进行爬取，但不可能漫无目的地去爬取，如果漫无目的地去爬取，那么就需要花费大量的时间，所以我们需要根据指定的主题，指定好对应的爬取策略与爬取路径，这样，才可以让我们在指定的时间内从海量的互联网信息中尽可能多的爬取出与主题相关的信息，而根据指定的主题来制定爬取策略与爬取路径是爬虫的定向爬取技术的重要组成部分。

3) 在 BeautifulSoup 中，进行信息筛选的方法策略主要有：通过正则表达式来筛选，通过 XPath 表达式来筛选，通过 CSS 表达式。

## 了解 Python 爬虫框架

在这一章中，我们会带大家了解 Python 的爬虫框架，包括什么是爬虫框架以及 Python 中常见的爬虫框架，让大家对 Python 爬虫框架有一个基本的认识。

### 10.1 什么是 Python 爬虫框架

简单来说，Python 的爬虫框架就是一些爬虫项目的半成品。比如可以将一些常见爬虫功能的实现代码部分写好，然后留下一些接口，在做不同的爬虫项目时，我们只需要根据实际情况，编写少量需要变动的代码部分，并按照需求调用这些接口，即可以实现一个爬虫项目。

所以，Python 的爬虫框架是一些爬虫项目的半成品，这里的“半成品”有两层含义：

1) 这些框架并不是爬虫项目的成品，需要用户根据具体爬虫任务更改之后才可以正常使用。

2) 在框架中已经实现了很多爬虫要实现的常见功能，所以能够让我们在使用框架开发爬虫项目的时候节省很多精力，从而更高效地开发出一些优质爬虫。

可以看到，爬虫框架主要就是将一些常见的功能代码、业务逻辑等进行封装，从而能够让我们以更高的效率开发出对应的爬虫项目。

### 10.2 常见的 Python 爬虫框架

在 Python 中，开源爬虫框架有很多，甚至我们也可以自己写一些爬虫框架。我们并不



需要掌握每一种开源的爬虫框架，只需要深入地掌握一种爬虫框架，对其他的爬虫框架有一些基本的了解即可。因为，在深入地掌握了一种爬虫框架以后，其他的爬虫框架也会很容易掌握，大部分爬虫框架实现的基本方式大同小异，而对其他的爬虫框架有一些基本的了解有助于我们拓宽知识面。

在本书后续章节中，我们会以 Python 爬虫框架中的 Scrapy 框架作为重点讲解内容，而对其他的爬虫框架我们仅需要简单了解即可。

Python 中常见的爬虫框架主要有：

- 1) Scrapy 框架
- 2) Crawley 框架
- 3) Portia 框架
- 4) newspaper 框架
- 5) python-goose 框架

接下来，我们会分别为大家介绍这 5 种常见的 Python 爬虫框架。

## 10.3 认识 Scrapy 框架

Scrapy 框架是一套比较成熟的 Python 爬虫框架，是使用 Python 开发的快速、高层次的信息爬取框架，可以高效率地爬取 Web 页面并提取出我们关注的结构化数据。

Scrapy 框架的应用领域有很多，比如网络爬虫开发、数据挖掘、数据监测、自动化测试等。

Scrapy 的官网地址是：<http://scrapy.org/>。图 10-1 是 Scrapy 的官网界面。



图 10-1 Scrapy 的官网界面

Scrapy 是一套开源的框架，开源也就意味着我们能够看到并且免费使用 Scrapy 的所有

代码。

在本节中，我们仅需要对 Scrapy 框架进行简单了解即可。

## 10.4 认识 Crawley 框架

Crawley 也是使用 Python 开发出来的一款爬虫框架，该框架致力于改变人们从互联网中提取数据的方式，让大家可以更高效地从互联网中爬取对应内容。

Crawley 的官网地址是：<http://project.crawley-cloud.com/>。图 10-2 为 Crawley 的官网界面。

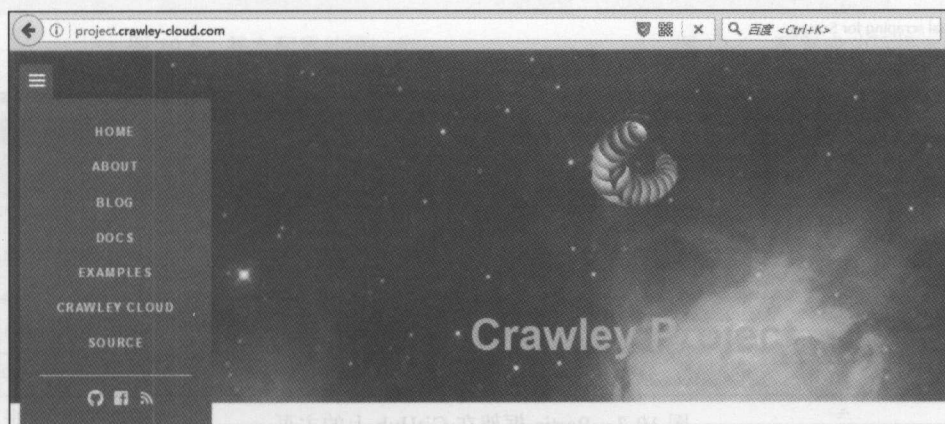


图 10-2 Crawley 的官网界面

我们可以在该网站中对 Crawley 进行相应了解并下载 Crawley 框架。

Crawley 框架的主要特点有：

- 1) 高速爬取对应网站内容。
- 2) 可以将爬取到的内容轻松地存储到关系型数据库中，比如 Postgres、MySQL、Oracle、SQLite 等数据库。
- 3) 可以将爬取到的数据导出为 JSON、XML 等格式。
- 4) 支持非关系型数据库，比如 MongoDB、CouchDB 等。
- 5) 支持使用命令行工具。
- 6) 可以使用你喜欢的工具提取数据，比如使用 XPath 或者 PyQuery 等工具。
- 7) 支持使用 Cookie 登录并访问那些只有登录才能够访问的网页。
- 8) 简单易学。

可以看到，Crawley 框架的功能相对来说还是较多的。对该框架有兴趣的读者，可以自行访问 Crawley 框架的官网并下载该框架进行相应探索。

需要掌握每一种开源的爬虫框架，只需要深入地掌握一种爬虫框架，对其他的爬虫框架有一些基本的了解即可。因为，在深入地掌握了一种爬虫框架以后，其他的爬虫框架也会很容易掌握，大部分爬虫框架实现的基本方式大同小异，而对其他的爬虫框架有一些基本的了解有助于我们拓宽知识面。

在本书后续章节中，我们会以 Python 爬虫框架中的 Scrapy 框架作为重点讲解内容，而对其他的爬虫框架我们仅需要简单了解即可。

Python 中常见的爬虫框架主要有：

- 1) Scrapy 框架
- 2) Crawlley 框架
- 3) Portia 框架
- 4) newspaper 框架
- 5) python-goose 框架

接下来，我们会分别为大家介绍这 5 种常见的 Python 爬虫框架。

## 10.3 认识 Scrapy 框架

Scrapy 框架是一套比较成熟的 Python 爬虫框架，是使用 Python 开发的快速、高层次的信息爬取框架，可以高效率地爬取 Web 页面并提取出我们关注的结构化数据。

Scrapy 框架的应用领域有很多，比如网络爬虫开发、数据挖掘、数据监测、自动化测试等。

Scrapy 的官网地址是：<http://scrapy.org/>。图 10-1 是 Scrapy 的官网界面。



图 10-1 Scrapy 的官网界面

Scrapy 是一套开源的框架，开源也就意味着我们能够看到并且免费使用 Scrapy 的所有

## 了解 Python 爬虫框架

在这一章中，我们会带大家了解 Python 的爬虫框架，包括什么是爬虫框架以及 Python 中常见的爬虫框架，让大家对 Python 爬虫框架有一个基本的认识。

### 10.1 什么是 Python 爬虫框架

简单来说，Python 的爬虫框架就是一些爬虫项目的半成品。比如可以将一些常见爬虫功能的实现代码部分写好，然后留下一些接口，在做不同的爬虫项目时，我们只需要根据实际情况，编写少量需要变动的代码部分，并按照需求调用这些接口，即可以实现一个爬虫项目。

所以，Python 的爬虫框架是一些爬虫项目的半成品，这里的“半成品”有两层含义：

1) 这些框架并不是爬虫项目的成品，需要用户根据具体爬虫任务更改之后才可以正常使用。

2) 在框架中已经实现了很多爬虫要实现的常见功能，所以能够让我们在使用框架开发爬虫项目的时候节省很多精力，从而更高效地开发出一些优质爬虫。

可以看到，爬虫框架主要就是将一些常见的功能代码、业务逻辑等进行封装，从而能够让我们以更高的效率开发出对应的爬虫项目。

### 10.2 常见的 Python 爬虫框架

在 Python 中，开源爬虫框架有很多，甚至我们也可以自己写一些爬虫框架。我们并不



开发内容，使用标准形式，开发小脚本

脚本名是：爬取代码

前面章节中，我们的爬虫项目都是一步步写出来的，使用纯手写的方式来实现这些项目，相对来说会慢一些，如果我们有一套开发相对完备的框架，那么写少量代码就可以实现一样甚至更复杂的功能。因为很多代码都已经事先在框架中封装好了，从而大大提高了项目开发的效率。在接下来的学习中，我们会重点介绍框架爬虫项目的开发。

脚本名是：爬虫

开发内容，使用小脚本开发爬虫

脚本名是：爬虫

开发内容，使用小脚本开发爬虫

同样，由于篇幅有限，“……”处省略了部分执行结果。可以看到，此时已经成功爬取了9页的评论信息，并且实现了自动加载新内容的功能。

通过这个事例的学习，相信大家对爬虫的定向爬取技术已经有了比较深入的了解，并且懂得了如何结合Fiddler来分析网页的会话信息，学会了如何在同一个页面中自动加载新内容的方法，希望可以抛砖引玉，让大家写出更多优质的网络爬虫。

## 9.4 小结

1) 通常来说，爬虫的定向爬取技术就是根据指定的主题，对需要爬取的网页中的内容进行筛选，比如我们可以使用正则表达式来匹配网页中的内容，并可以指定爬取的内容范围，从而实现对网页内容的定向爬取。

2) 互联网的信息是海量的，我们在一个相对短的时间内，不可能把所有的信息都爬取到我们感兴趣的信息，也不可能漫无目的地去爬取，如果漫无目的地爬取，那必然会浪费大量的时间，所以我们需要根据指定的主题，制定出对应的爬取策略与爬取规则，这样，才可以让我们在较短的时间内从海量的互联网信息中尽可能多的爬取出与主题相关的信息。而根据指定的主题制定对应的爬取策略与爬取规则是爬虫的定向爬取技术的核心与重点部分。

3) 在Python中，进行信息筛选的方法策略主要有：使用正则表达式筛选，通过XPath表达式筛选，通过Xpath表达式。

### 第三篇 *Part 3*

## 框架实现篇

在这一章中，我们会带大家了解 Python 的爬虫框架，包括什么爬虫框架以及 Python 中常见的爬虫框架，让大家对 Python 爬虫框架有一个基本的认识。

### 10.1 什么是 Python 爬虫框架

简单来说，Python 的爬虫框架就是一些爬虫项目的半成品。比如，我们可以将一些常见爬虫功能的实现代码部分写好，然后留下一些接口，在做不同的爬虫项目时，我们只需要根据实际情况，编写少量代码，调用这些接口，即可以实现一个爬虫项目。

- 第 10 章 了解 Python 爬虫框架
- 第 11 章 爬虫利器——Scrapy 安装与配置
- 第 12 章 开启 Scrapy 爬虫项目之旅
- 第 13 章 Scrapy 核心架构
- 第 14 章 Scrapy 中文输出与存储
- 第 15 章 编写自动爬取网页的爬虫
- 第 16 章 CrawlSpider
- 第 17 章 Scrapy 高级应用

### 10.2 常见的 Python 爬虫框架

在 Python 中，开源爬虫框架有很多，甚至我们也可以自己写一些爬虫框架。我们并不

评论内容是：没有你看怒火 v 很符合国家

用户名是：X 秒殺于無形

评论内容是：这，怎么

用户名是：小二，来碗泪流满面

评论内容是：剧本里就是下雪的呀

.....

第 9 页评论内容

用户名是：----

评论内容是：明明说

用户名是：后来、

评论内容是：这几个男演员真是丑到家

用户名是：捷

评论内容是：对

.....

同样，由于篇幅有限，“.....”处省略了部分执行结果。可以看到，此时已经成功爬取了 9 页的评论信息，并且实现了自动加载新内容的功能。

通过这个案例的学习，相信大家对爬虫的定向爬取技术已经有了比较深入的了解，并且懂得了如何结合 Fiddler 来分析网页的会话信息，学会了如何在同一个页面中自动加载新内容的方法，希望可以抛砖引玉，让大家写出更多优质的网络爬虫。

## 9.4 小结

1) 通俗来说，爬虫的定向爬取技术就是根据设置的主题，对要爬取的网址或者网页中的内容进行筛选，比如我们可以使用正则表达式进行筛选等，筛选之后，再爬取对应的网址中的内容，并可以根据爬取到的内容再次进行筛选。

2) 互联网的信息是海量的，我们在一个相对较短的时间内要尽可能多的爬取到我们感兴趣的信息，则不可能漫无目的地去爬取，如果漫无目的地去爬取，则必然会浪费大量的时间，所以我们需要根据设置的主题，拟定出对应的爬取策略与爬取规则，这样，才可以让我们在较短的时间内从海量的互联网信息中尽可能多的爬取出与主题相关的信息。而根据设定的主题建立爬虫的爬取策略与爬取规则是爬虫的定向爬取技术的核心与重点部分。

3) 在 Python 中，进行信息筛选的方法策略主要有：通过正则表达式筛选、通过 XPath 表达式筛选、通过 xslt 筛选。

```

cjar=http.cookiejar.CookieJar()
opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cjar))
headall=[]
for key,value in headers.items():
    item=(key,value)
    headall.append(item)
opener.addheaders = headall
urllib.request.install_opener(opener)
# 建立一个自定义函数 crawl(vid,comid), 实现自动爬取对应评论网页并返回爬取数据
def crawl(vid,comid):
    url= "http://coral.qq.com/article/"+vid+"/comment?commentid="+comid+"&regnum=20"
    data=urllib.request.urlopen(url).read().decode("utf-8")
    return data
idpat='"id": "(.*)"'
userpat='"nick": "(.*)"'
conpat='"content": "(.*)"'
# 第一层循环, 代表爬取多少页, 每一次外层循环爬取一页
for i in range(1,10):
    print("-----")
    print("第 "+str(i)+" 页评论内容 ")
    data=crawl(vid,comid)
    # 第二层循环, 根据爬取的结果提取并处理每条评论的信息, 一页 20 条评论
    for j in range(0,20):
        idlist=re.compile(idpat,re.S).findall(data)
        userlist=re.compile(userpat,re.S).findall(data)
        conlist=re.compile(conpat,re.S).findall(data)
        print("用户名是 :"+eval('u'+userlist[j]+''))
        print("评论内容是 :"+eval('u'+conlist[j]+''))
        print("\n")
    # 将 comid 改变为该页的最后一条评论 id, 实现不断自动加载
    comid=idlist[19]

```

该程序同样会伪装成浏览器对对应评论页面进行爬取, 此时, 我们将爬取的过程封装成了一个函数, 这样就可以根据不同的参数传递, 从而实现爬取不同的评论网页。同时, 进行了两层 for 循环, 第一层 for 循环实现爬行页数的控制, 每次循环爬取一页, 第二层 for 循环实现对每页的评论进行分别输出处理, 每次循环输出一条评论信息。完成第二层 for 循环后, 需要在第一层 for 循环的末尾处对 comid 进行重新赋值, 从而实现新评论的加载。

程序的执行结果如下:

```
>>>
```

```
===== RESTART: D:\Python35\9.3.py =====
```

```
第 1 页评论内容
```

```
用户名是: 心如薄荷天然凉 i
```

```
评论内容是: 这个问题问得好
```

```
用户名是: `颖~
```



该程序会伪装成浏览器对相关评论页面进行爬取，爬取后会通过正则表达式将对应信息提取出来，然后通过 for 循环将每条评论信息分别输出。

执行结果如下所示，由于篇幅所限，获取的评论内容较多，在“……”处省略了部分执行结果：

```
>>>
===== RESTART: D:\Python35\9.3.py =====
用户名是：心如薄荷天然凉 i
评论内容是：这个问题问得好

用户名是：`颖~
评论内容是：没有你看怒火 v 很符合国家

用户名是：X秒殺于無形
评论内容是：这，怎么

用户名是：小二，来碗泪流满面
评论内容是：剧本里就是下雪的呀
.....
```

这个程序有一个缺点，就是无法加载新的评论。那么应当怎样实现加载新评论的功能呢？

我们知道，真实评论请求网址中的 commentid 字段代表的是起始评论的 ID。所以此时，我们只需要将每次评论页面中的最后一个评论 ID 提取出来，并赋值到 URL 网址中的对应位置，即可构造出新的 URL 网址，从而加载出新的评论。

所以，我们可以对上述代码进行相应改进，即可实现自动加载新评论的功能。

改进后的参考代码如下，代码中关键位置给出了注释。

```
import urllib.request
import http.cookiejar
import re
# 视频编号
vid="1472528692"
# 刚开始时候的评论 ID
comid="6173403130078248384"
url= "http:// coral.qq.com/article/" +vid+ "/comment?commentid="+comid+"&reqnum=20"
headers={ "Accept": " text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
           "Accept-Encoding": " gb2312,utf-8",
           "Accept-Language": " zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
           "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) Apple
WebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/
537.36 SE 2.X MetaSr 1.0","Connection": "keep-alive",
           "referer": "qq.com" }
```

```

cjar=http.cookiejar.CookieJar()
opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cjar))
headall=[]
for key,value in headers.items():
    item=(key,value)
    headall.append(item)
opener.addheaders = headall
urllib.request.install_opener(opener)
# 建立一个自定义函数 crawl(vid,comid), 实现自动爬取对应评论网页并返回爬取数据
def crawl(vid,comid):
    url= "http://coral.qq.com/article/"+vid+"/comment?commentid="+comid+"&reqnum=20"
    data=urllib.request.urlopen(url).read().decode("utf-8")
    return data
idpat="id": "(.*)"
userpat="nick": "(.*)",
conpat="content": "(.*)",
# 第一层循环, 代表爬取多少页, 每一次外层循环爬取一页
for i in range(1,10):
    print("-----")
    print("第 "+str(i)+" 页评论内容 ")
    data=crawl(vid,comid)
    # 第二层循环, 根据爬取的结果提取并处理每条评论的信息, 一页 20 条评论
    for j in range(0,20):
        idlist=re.compile(idpat,re.S).findall(data)
        userlist=re.compile(userpat,re.S).findall(data)
        conlist=re.compile(conpat,re.S).findall(data)
        print(" 用户名是 :"+eval('u'+userlist[j]+''))
        print(" 评论内容是 :"+eval('u'+conlist[j]+''))
        print("\n")
    # 将 comid 改变为该页的最后一条评论 id, 实现不断自动加载
    comid=idlist[19]

```

该程序同样会伪装成浏览器对对应评论页面进行爬取, 此时, 我们将爬取的过程封装成了一个函数, 这样就可以根据不同的参数传递, 从而实现爬取不同的评论网页。同时, 进行了两层 for 循环, 第一层 for 循环实现爬行页数的控制, 每次循环爬取一页, 第二层 for 循环实现对每页的评论进行分别输出处理, 每次循环输出一条评论信息。完成第二层 for 循环后, 需要在第一层 for 循环的末尾处对 comid 进行重新赋值, 从而实现新评论的加载。

程序的执行结果如下:

```

>>>
===== RESTART: D:\Python35\9.3.py =====

```

第 1 页评论内容

用户名是: 心如薄荷天然凉 i

评论内容是: 这个问题问得好

用户名是: `颖~

评论内容是：没有你看怒火 v 很符合国家

用户名是：X秒殺于無形

评论内容是：这，怎么

用户名是：小二，来碗泪流满面

评论内容是：剧本里就是下雪的呀

.....

第 9 页评论内容

用户名是：----

评论内容是：明明说

用户名是：后来、

评论内容是：这几个男演员真是丑到家

用户名是：捷

评论内容是：对

.....

同样，由于篇幅有限，“.....”处省略了部分执行结果。可以看到，此时已经成功爬取了 9 页的评论信息，并且实现了自动加载新内容的功能。

通过这个案例的学习，相信大家对爬虫的定向爬取技术已经有了比较深入的了解，并且懂得了如何结合 Fiddler 来分析网页的会话信息，学会了如何在同一个页面中自动加载新内容的方法，希望可以抛砖引玉，让大家写出更多优质的网络爬虫。

## 9.4 小结

1) 通俗来说，爬虫的定向爬取技术就是根据设置的主题，对要爬取的网址或者网页中的内容进行筛选，比如我们可以使用正则表达式进行筛选等，筛选之后，再爬取对应的网址中的内容，并可以根据爬取到的内容再次进行筛选。

2) 互联网的信息是海量的，我们在一个相对较短的时间内要尽可能多的爬取到我们感兴趣的信息，则不可能漫无目的地去爬取，如果漫无目的地去爬取，则必然会浪费大量的时间，所以我们需要根据设置的主题，拟定出对应的爬取策略与爬取规则，这样，才可以让我们在较短的时间内从海量的互联网信息中尽可能多的爬取出与主题相关的信息。而根据设定的主题建立爬虫的爬取策略与爬取规则是爬虫的定向爬取技术的核心与重点部分。

3) 在 Python 中，进行信息筛选的方法策略主要有：通过正则表达式筛选、通过 XPath 表达式筛选、通过 xslt 筛选。

### 第三篇 *Part 3*

## 框架实现篇

在这一章中，我们会带大家了解 Python 的爬虫框架，包括什么爬虫框架以及 Python 中常见的爬虫框架，让大家对 Python 爬虫框架有一个基本的认识。

### 10.1 什么是 Python 爬虫框架

简单来说，Python 的爬虫框架就是一些爬虫项目的半成品。比如，我们可以将一些常见爬虫功能的实现代码拆分写好，然后留下一些接口，在做不同的爬虫项目时，我们只需要根据实际情况，编写少量代码，然后调用这些接口，即可以实现一个爬虫项目。

- 第 10 章 了解 Python 爬虫框架
- 第 11 章 爬虫利器——Scrapy 安装与配置
- 第 12 章 开启 Scrapy 爬虫项目之旅
- 第 13 章 Scrapy 核心架构
- 第 14 章 Scrapy 中文输出与存储
- 第 15 章 编写自动爬取网页的爬虫
- 第 16 章 CrawlSpider
- 第 17 章 Scrapy 高级应用

### 10.2 常见的 Python 爬虫框架

在 Python 中，开源爬虫框架有很多，甚至我们也可以自己写一些爬虫框架。我们并不



该地方就是爬虫程序需要爬取网页的URL。

程序代码如下：

前面章节中，我们的爬虫项目都是一步步写出来的，使用纯手写的方式来实现这些项目，相对来说会慢一些，如果我们有一套开发相对完备的框架，那么写少量代码就可以实现一样甚至更复杂的功能。因为很多代码都已经事先在框架中封装好了，从而大大提高了项目开发的效率。在接下来的学习中，我们会重点介绍框架爬虫项目的开发。

程序代码如下：

该地方就是爬虫程序需要爬取网页的URL。

程序代码如下：

该地方就是爬虫程序需要爬取网页的URL。

同样，由于篇幅有限，“……”处省略了部分执行结果，可以看到，此时已经成功爬取了9页的评论信息，并且实现了自动加载新内容的功能。

通过这个案例的学习，相信大家对于爬虫的定向爬取技术已经有了比较深入的了解，并且懂得了如何结合XPath来分析网页的会话信息，学会了如何在同一个页面中自动加载新内容的方法，希望可以抛砖引玉，让大家写出更多优秀的网络爬虫。

## 9.4 小结

1) 总的来说，爬虫的定向爬取技术就是根据网页的URL，对要爬取的网址或者网页中的内容进行筛选，比如我们可以使用正则表达式来匹配URL，爬取对应的网址中的内容，并可以根据爬取到的内容进一步筛选出我们需要的信息。

2) 互联网的信息是海量的，我们不可能在一个很短的时间内爬取到我们感兴趣的信息，也不可能漫无目的地去爬取，如果那样不仅会浪费大量的时间，所以我们需要根据设置的主题，指定爬取对应的URL或者URL中的内容，这样才能让我们在较短的时间内从海量的互联网信息中尽可能多地爬取出与主题相关的信息。而根据设置的主题来指定爬取策略与爬取规则是爬虫的定向爬取技术的重要组成部分。

3) 在Python中，进行信息筛选的方法策略主要有：通过正则表达式筛选，通过XPath

## 了解 Python 爬虫框架

在这一章中，我们会带大家了解 Python 的爬虫框架，包括什么是爬虫框架以及 Python 中常见的爬虫框架，让大家对 Python 爬虫框架有一个基本的认识。

### 10.1 什么是 Python 爬虫框架

简单来说，Python 的爬虫框架就是一些爬虫项目的半成品。比如可以将一些常见爬虫功能的实现代码部分写好，然后留下一些接口，在做不同的爬虫项目时，我们只需要根据实际情况，编写少量需要变动的代码部分，并按照需求调用这些接口，即可以实现一个爬虫项目。

所以，Python 的爬虫框架就是一些爬虫项目的半成品，这里的“半成品”有两层含义：

1) 这些框架并不是爬虫项目的成品，需要用户根据具体爬虫任务更改之后才可以正常使用。

2) 在框架中已经实现了很多爬虫要实现的常见功能，所以能够让我们在使用框架开发爬虫项目的时候节省很多精力，从而更高效地开发出一些优质爬虫。

可以看到，爬虫框架主要就是将一些常见的功能代码、业务逻辑等进行封装，从而能够让我们以更高的效率开发出对应的爬虫项目。

### 10.2 常见的 Python 爬虫框架

在 Python 中，开源爬虫框架有很多，甚至我们也可以自己写一些爬虫框架。我们并不

需要掌握每一种开源的爬虫框架，只需要深入地掌握一种爬虫框架，对其他的爬虫框架有一些基本的了解即可。因为，在深入地掌握了一种爬虫框架以后，其他的爬虫框架也会很容易掌握，大部分爬虫框架实现的基本方式大同小异，而对其他的爬虫框架有一些基本的了解有助于我们拓宽知识面。

在本书后续章节中，我们会以 Python 爬虫框架中的 Scrapy 框架作为重点讲解内容，而对其他的爬虫框架我们仅需要简单了解即可。

Python 中常见的爬虫框架主要有：

- 1) Scrapy 框架
- 2) Crawlley 框架
- 3) Portia 框架
- 4) newspaper 框架
- 5) python-goose 框架

接下来，我们会分别为大家介绍这 5 种常见的 Python 爬虫框架。

## 10.3 认识 Scrapy 框架

Scrapy 框架是一套比较成熟的 Python 爬虫框架，是使用 Python 开发的快速、高层次的信息爬取框架，可以高效率地爬取 Web 页面并提取出我们关注的结构化数据。

Scrapy 框架的应用领域有很多，比如网络爬虫开发、数据挖掘、数据监测、自动化测试等。

Scrapy 的官网地址是：<http://scrapy.org/>。图 10-1 是 Scrapy 的官网界面。



图 10-1 Scrapy 的官网界面

Scrapy 是一套开源的框架，开源也就意味着我们能够看到并且免费使用 Scrapy 的所有

代码。

在本节中，我们仅需要对 Scrapy 框架进行简单了解即可。

## 10.4 认识 Crawley 框架

Crawley 也是使用 Python 开发出来的一款爬虫框架，该框架致力于改变人们从互联网中提取数据的方式，让大家可以更高效地从互联网中爬取对应内容。

Crawley 的官网地址是：<http://project.crawley-cloud.com/>。图 10-2 为 Crawley 的官网界面。

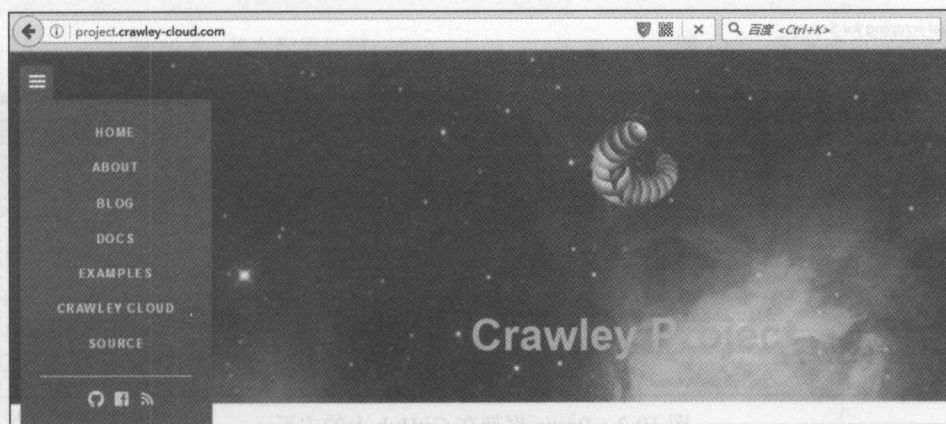


图 10-2 Crawley 的官网界面

我们可以在该网站中对 Crawley 进行相应了解并下载 Crawley 框架。

Crawley 框架的主要特点有：

- 1) 高速爬取对应网站内容。
- 2) 可以将爬取到的内容轻松地存储到关系型数据库中，比如 Postgres、MySQL、Oracle、SQLite 等数据库。
- 3) 可以将爬取到的数据导出为 JSON、XML 等格式。
- 4) 支持非关系型数据库，比如 MongoDB、CouchDB 等。
- 5) 支持使用命令行工具。
- 6) 可以使用你喜欢的工具提取数据，比如使用 XPath 或者 PyQuery 等工具。
- 7) 支持使用 Cookie 登录并访问那些只有登录才能够访问的网页。
- 8) 简单易学。

可以看到，Crawley 框架的功能相对来说还是较多的。对该框架有兴趣的读者，可以自行访问 Crawley 框架的官网并下载该框架进行相应探索。



## 10.5 认识 Portia 框架

Portia 框架是一款允许没有任何编程基础的用户可视化地爬取网页的爬虫框架。给出你要爬取的网页中感兴趣的数据内容,通过 Portia 框架,可以将你所要爬取的信息从相似的网页中自动提取出来。

如果我们要获取该框架,可以通过 Portia 在 GitHub 上的主页进行获取。

Portia 在 GitHub 上的主页地址是: <https://github.com/scrapinghub/portia/>。

如图 10-3 所示,为该框架在 GitHub 上的主页。

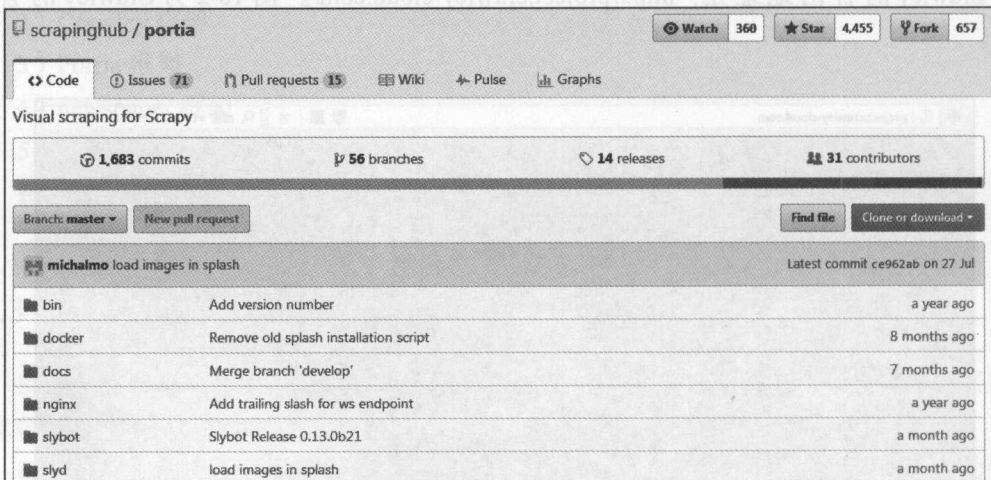


图 10-3 Portia 框架在 GitHub 上的主页

使用 Portia 框架两种方式,你可以将 Portia 框架下载到本地使用,也可以直接使用网页版的 Portia 框架。

如果你想快速了解 Portia 框架,可以直接使用网页版 Portia 框架,此时不需要将该框架下载到本地,只需要注册一个账号即可。

接下来我们就具体使用该框架的网页版从而快速了解 Portia 框架。

首先打开 Portia 框架的网页版地址: <https://portia.scrapinghub.com/>。

打开后如图 10-4 所示。

此时,我们需要注册一个账号并登录。登录后,需要进行基本配置,图 10-5 为基本配置的第一步。



图 10-4 Scrapinghub 登录界面

配置好之后，会出现如图 10-6 所示的提示信息。

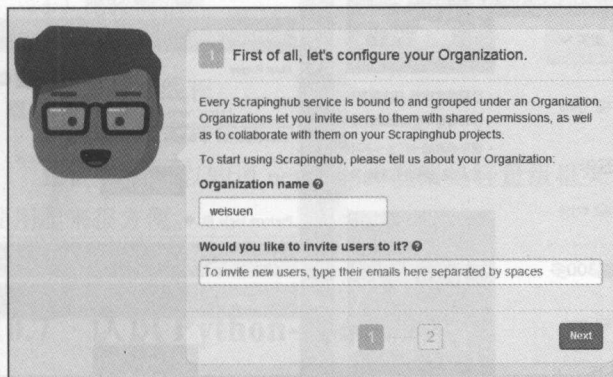


图 10-5 基本配置的第一步

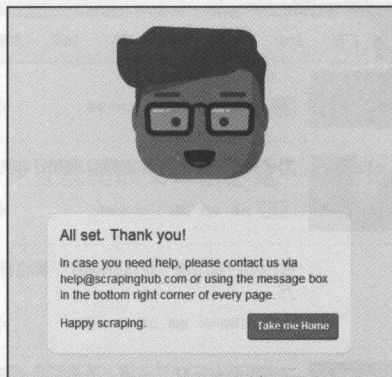


图 10-6 配置成功后提示界面

随后，单击“Take me Home”会出现如图 10-7 所示界面，此时我们就可以使用 Portia 框架进行网站的爬取了。

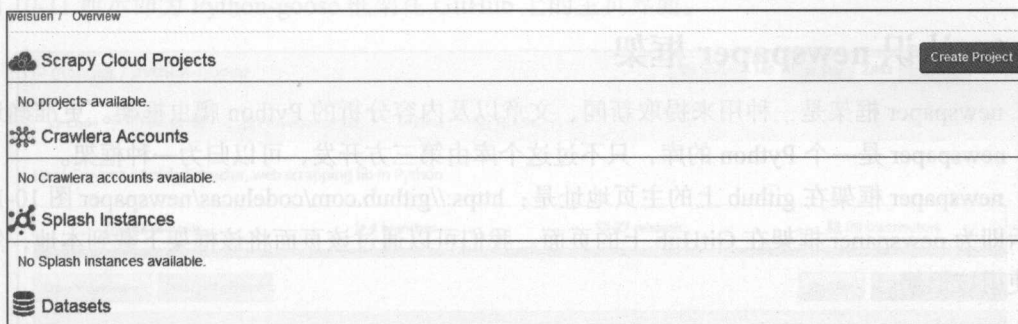


图 10-7 Home 页

首先，我们需要创建一个爬虫项目，单击“Create Project”会出现如图 10-8 所示界面。

此时我们可以设置对应的爬虫名称，并选择基于什么创建，比如可以选择基于 Portia 创建，也可以选择基于 Scrapy 创建。设置好后可以单击“Create”创建一个爬虫项目。

我们可以设置爬虫项目的相关信息，比如设置要爬取的网址、设置 Crawling 信息、设置 Samples 信息等。设置好后，即可以单击“Test spider”测试该爬虫，如图 10-9 所示，我们可以对网易新闻页面进行相应爬取。

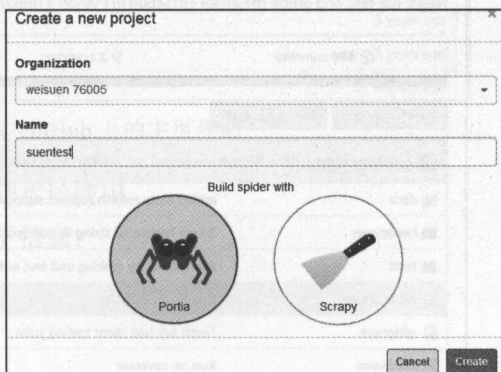


图 10-8 创建爬虫项目页面



图 10-9 爬虫设置界面

可以看到，使用 Portia 框架可以通过可视化的方式很方便地配置对应的爬虫，并实现爬虫项目。

## 10.6 认识 newspaper 框架

newspaper 框架是一种用来提取新闻、文章以及内容分析的 Python 爬虫框架。更准确地说，newspaper 是一个 Python 的库，只不过这个库由第三方开发，可以归为一种框架。

newspaper 框架在 github 上的主页地址是：<https://github.com/codelucas/newspaper> 图 10-10 所示即为 newspaper 框架在 GitHub 上的页面。我们可以通过该页面将该框架下载到本地，然后使用该框架。

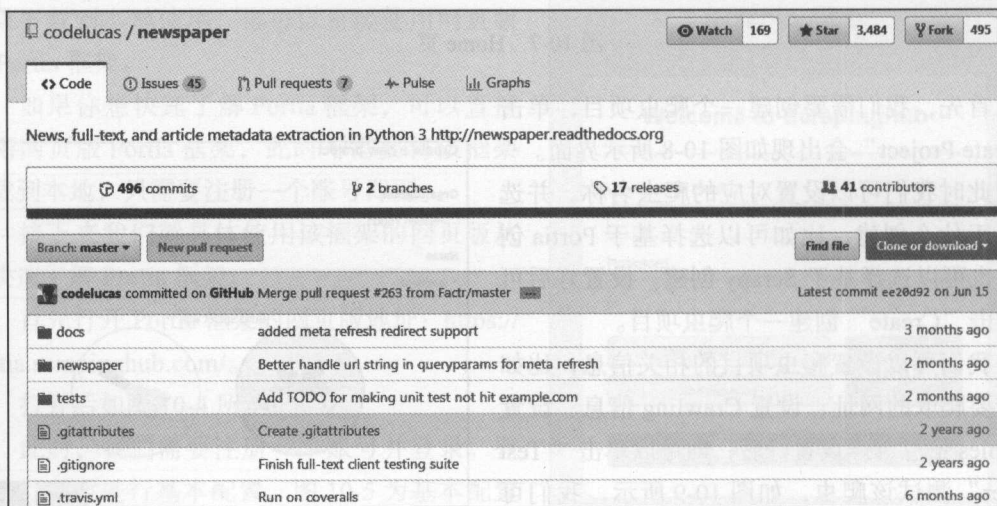


图 10-10 newspaper 框架在 GitHub 上的页面

newspaper 框架的主要特点有：

- 1) 比较简洁
- 2) 速度较快
- 3) 支持多线程
- 4) 支持十多种语言

由此我们可以知道 newspaper 框架是轻量级框架，并且就爬取文章信息这一功能来说，使用起来很方便。

## 10.7 认识 Python-goose 框架

Goose 本来是一款用 Java 写的文章提取工具，Xavier Grangier 用 Python 重写了 Goose，并将重写后的 Goose 命名为 Python-goose。

所以，Python-goose 框架实现的功能同样是进行文章提取。

Python-goose 框架在 GitHub 上的主页地址是：<https://github.com/grangier/python-goose>

图 10-11 所示即为 Python-goose 框架在 GitHub 上的主页界面。

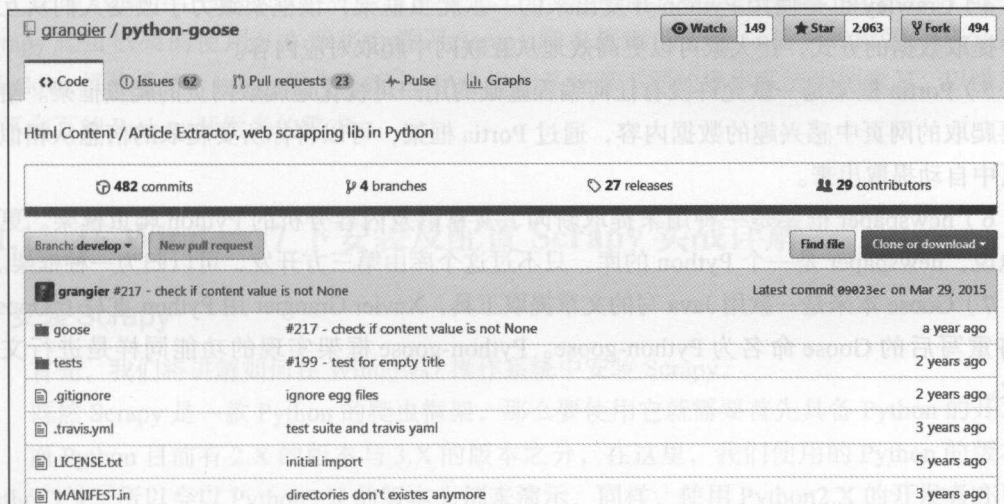


图 10-11 Python-goose 框架在 GitHub 上的主页界面

通过该网页，可以将 Python-goose 框架下载到本地使用。

我们可以使用 Python-goose 框架主要提取如下信息：

- 1) 文章主体内容
- 2) 元描述
- 3) 元标签
- 4) 文章中的任何 YouTube/Vimeo 视频



### 5) 文章中的主要图片

使用 Python-goose 框架可以很方便地满足这些需求,感兴趣的读者可以通过我们上面提供的 GitHub 上 Python-goose 框架的主页地址将该框架下载到本地,并探索如何使用其爬取相应的文章信息。

## 10.8 小结

1) 简单来说,Python 的爬虫框架就是一些爬虫项目的半成品。比如我们可以将一些常见爬虫功能的实现代码部分写好,然后留下一些接口,在做不同的爬虫项目时,我们只需要根据实际情况,手写少量需要变动的代码部分,并按照需求调用这些接口,即可以实现一个爬虫项目。

2) Scrapy 框架是一套比较成熟的 Python 爬虫框架,是使用 Python 开发的快速、高层次的信息爬取框架,可以高效地爬取 Web 页面并提取出我们关注的结构化数据。

3) Scrapy 框架的应用领域有很多,比如网络爬虫开发、数据挖掘、数据监测、自动化测试等。

4) Crawley 也是使用 Python 开发出来的一款爬虫框架,该框架致力于改变人们从互联网中提取数据的方式,让大家可以更高效地从互联网中爬取对应内容。

5) Portia 框架是一款允许没有任何编程基础的用户可视化地爬取网页的爬虫框架。给出你要爬取的网页中感兴趣的数据内容,通过 Portia 框架,可以将你所要爬取的信息从相似的网页中自动提取出来。

6) newspaper 框架是一种用来提取新闻、文章以及内容分析的 Python 爬虫框架。更准确地说,newsaper 是一个 Python 的库,只不过这个库由第三方开发,可以归为一种框架。

7) Goose 本来是一款用 Java 写的文章提取工具,Xavier Grangier 用 Python 重写了 Goose,并将重写后的 Goose 命名为 Python-goose。Python-goose 框架实现的功能同样是进行文章提取。

## 爬虫利器——Scrapy 安装与配置

图 11-1 安装后的 Fiddler 界面

在第 10 章中，我们已经对爬虫框架进行了基本的了解，在本章，我们将开始学习 Scrapy 爬虫框架的使用。本章将会学习 Scrapy 爬虫框架的安装与基本的配置，并分别以 Windows 操作系统、Linux 操作系统、MAC 操作系统等不同的操作系统为例学习，以满足不同平台下的 Python 开发者的需求。

### 11.1 在 Windows7 下安装及配置 Scrapy 实战详解

#### 1. 安装 Scrapy

首先，我们将讲解如何在 Windows7 操作系统中安装 Scrapy。

既然 Scrapy 是一款 Python 的爬虫框架，那么要使用它就需要首先具备 Python 的开发环境，而 Python 目前有 2.X 的版本与 3.X 的版本之分，在这里，我们使用的 Python 的版本为 Python3.X，所以会以 Python3.X 的版本为例来演示，同样，使用 Python2.X 的开发者亦可以参照。

由于我们之前使用了 Fiddler 作为代理服务器进行调试分析，所以有可能你的客户端也会默认使用该代理服务器，而在 Scrapy 的安装过程中，我们不需要用到该软件，为了避免该软件的影响，可以对该软件进行相应的设置。

首先，打开 Fiddler，然后依次进入“Tools → Fiddler Options → Connections”，此时我们可以看到如图 11-1 所示界面。

可以发现，此时“Act as system proxy on startup”选项与“Monitor all connections”选项都是勾选状态，“Act as system proxy on startup”选项表示在启动时让 Fiddler 作为系统的

代理,“Monitor all connections”选项表示让 Fiddler 监控所有的连接,所以此时,我们可以取消这两个选项的勾选状态,如图 11-2 所示。

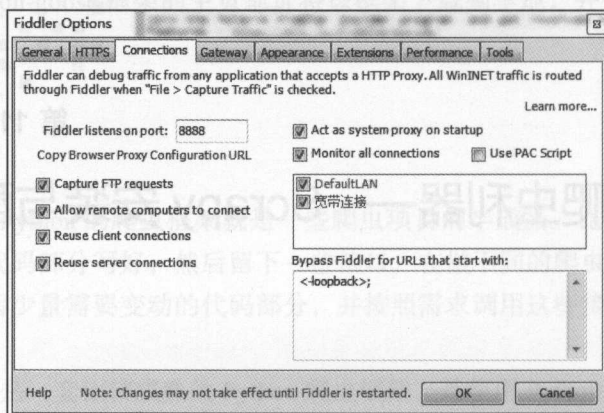


图 11-1 进入“Tools → Fiddler Options → Connections”

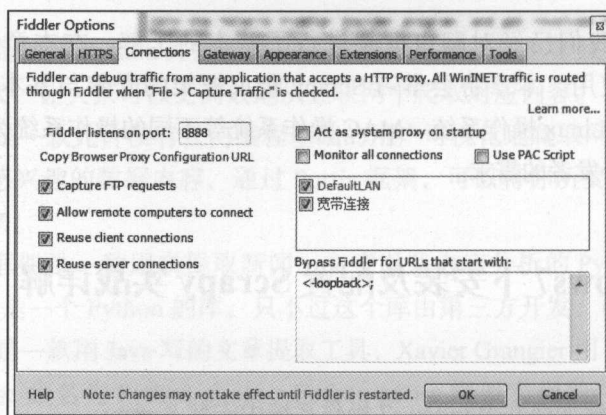


图 11-2 取消 Fiddler 中的对应选项

取消了勾选状态之后,可以看到,“DefaultLAN”与“宽带连接”还是勾选状态,如图 11-2 所示,此时,我们重启一下 Fiddler,重启后再进入“Tools → Fiddler Options → Connections”,随后可以看到“DefaultLAN”与“宽带连接”这两项的勾选状态已经取消,如图 11-3 所示,此时, Fiddler 配置完成,这样 Fiddler 这款软件就不会干扰客户端的连接了,在我们需要用的调试分析的时候,再重新勾选对应选项即可。

解决了 Fiddler 之后,就可以正式进入 Scrapy 的安装了。

我们可以使用 pip 来安装 Scrapy。

在 Python3 中, pip 是默认安装好的,在 windows 中“开始”的输入框处输入 cmd,进入 cmd 命令行模式。

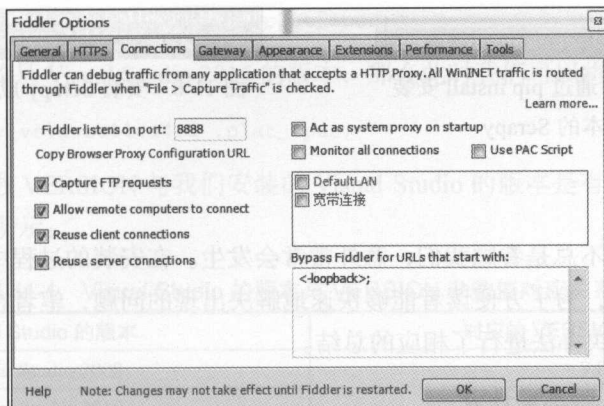


图 11-3 重启后的 Fiddler 对应界面

进入 cmd 模式之后，我们可以先测试一下 pip 是否已经安装，输入指令：

```
pip -h
```

如果能出现如图 11-4 所示的信息，说明 pip 已经安装好了，我们可以看到 pip 的一些指令提示信息。

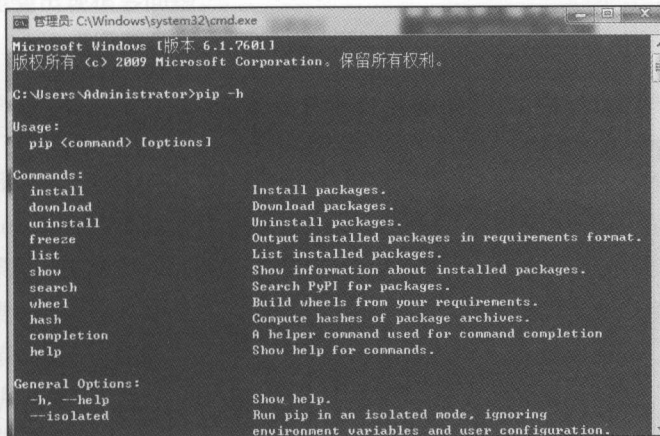


图 11-4 运行 pip -h 出现的界面

随后，可以使用 `pip install scrapy==1.1.0rc3` 指令安装对应版本的 Scrapy，如图 11-5 所示。

在此，我们指定了 Scrapy 的版本，当然，你也可以不指定版本，直接使用 `pip install Scrapy` 指令来安装 Scrapy，目前新版的 Scrapy 已经支持 Python3 了。

如果不出意外，通过这个指令就可以完成 Scrapy 的安装。

安装完成之后，会出现如图 11-6 所示信息。



```
C:\>pip install scrapy==1.1.0rc3
```

图 11-5 cmd 下通过 pip install 安装  
对应版本的 Scrapy

```
Installing collected packages: parsel, scrapy  
Successfully installed parsel-1.0.3 scrapy-1.1.0rc3
```

图 11-6 安装 Scrapy 成功后的提示信息

## 2. 常见问题

但是幸运之神并不总是眷顾我们，意外常常会发生。在安装的过程中，免不了会出现各种类型的问题，在此，为了方便读者能够快速解决出现的问题，笔者在此对在安装的过程中常出现的问题与解决办法进行了相应的总结。

### 常见问题 1: pip 版本需升级

如果你的 pip 版本较老，可能在安装的过程中需要更新对应的 pip 版本。

所以，此时你可能会出现如图 11-7 所示的问题。

```
You are using pip version 8.1.1, however version 8.1.2 is available.  
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

图 11-7 pip 需升级提示信息

出现这个问题之后，还可以通过 `python -m pip install --upgrade pip` 指令来完成 pip 的升级，如图 11-8 所示。

```
C:\Users\Administrator>python -m pip install --upgrade pip
```

图 11-8 cmd 下升级 pip

升级完成之后，这一类错误即可解决。

### 常见问题 2: unable to find vcvarsall.bat

在安装的过程中，有可能会出现“无法找到 `vcvarsall.bat`”等错误提示，如图 11-9 所示。

```
running build_ext  
building 'lxml.etree' extension  
error: Unable to find vcvarsall.bat  
  
-----  
Command "D:\python35\python.exe -u -c "import setuptools; tokenize = __file__ = 'C:\Users\Administrator\AppData\Local\Temp\pip-build-W86pige\lxml\setup.py'; exec(compile(getattr(tokenize, 'open', open)(__file__).read().replace('\r\n', '\n'), __file__, 'exec'))" install --record C:\Users\Administrator\AppData\Local\Temp\pip-install-record\install-record.txt --log C:\Users\Administrator\AppData\Local\Temp\pip-install-record\install-record.txt failed with error code 1 in C:\Users\Administrator\AppData\Local\Temp\pip-build-W86pige\lxml\
```

图 11-9 出现“无法找到 `vcvarsall.bat`”等错误提示

如果出现这个错误，我们可以安装对应版本的 Visual Studio 解决。那么对应的版本到底是什么版本呢？

可以根据 Python 的一个文件进行设置，我们打开 Python 安装目录下的“`Lib/distutils/msvc9compiler.py`”文件，比如，笔者的 Python 安装目录是“`D:\Python35`”，所以此时我们可以打开“`D:\Python35\Lib\distutils\msvc9compiler.py`”文件，打开该文件后，可以看到类似如下的代码：

```
vc_env = query_vcvarsall(VERSION, plat_spec)
```

如果我们安装的是 Visual Studio 2015 的版本，那么此时我们可以将这一行代码改为：

```
vc_env = query_vcvarsall(14.0, plat_spec)
```

可以看到，参数 VERSION 与我们安装的 Visual Studio 的版本是有对应关系的，具体的对应关系如表 11-1 所示：

表 11-1 Visual Studio 的版本与 VERSION 参数值对应关系表

Visual Studio 的版本	对应的 VERSION 参数值
Visual Studio 2008	9.0
Visual Studio 2010	10.0
Visual Studio 2012	11.0
Visual Studio 2013	12.0
Visual Studio 2014	13.0
Visual Studio 2015	14.0

设置好参数之后，我们就可以下载对应的 Visual Studio 版本并进行安装，在此我们下载的 Visual Studio 版本是 Visual Studio 2015，建议大家使用专业版，因为社区版（Community）在安装的时候可能会出现很多问题。

Visual Studio 2015 专业版的官方下载地址是：

```
http://download.microsoft.com/download/B/8/9/B898E46E-CBAE-4045-A8E2-2D33DD36F3C4/vs2015.pro_chs.iso
```

下载之后，我们就可以解压并进行安装。

安装 Visual Studio，需要对应版本的 .NET Framework 的支持，如果没有安装对应版本的 .NET Framework，可能会出现如图 11-10 所示的错误界面：

如果出现该错误界面，我们可以根据错误提示信息，下载对应版本的 Microsoft NET Framework 安装即可解决该错误。

在安装好 Visual Studio 之后，再使用 pip 安装 Scrapy 就不会出现该错误了。

### 常见问题 3：缺少 lxml

如果缺少 lxml，我们会发现出现如图 11-11 所示的错误：

此时，若要解决该错误，就需要安装对应的 lxml，我们可以先下载 lxml 的 whl 格式的文件，

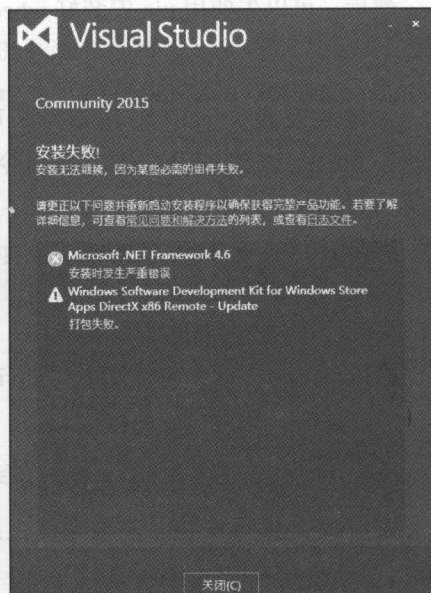


图 11-10 需要 .NET Framework 的提示界面

然后使用 pip 进行安装。

```

xmlPathInitjlo76w1.c
C:\Users\ADMINI~1\AppData\Local\Temp\xmlPathInitjlo76w1.c(1): fatal er
ror C1083: 无法打开包括文件: "libxml\xpath.h": No such file or directory
*****
Could not find function xmlCheckVersion in library libxml2. Is libxml2 insta
lled?
*****
error: command 'E:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\BIN\x86_and64\xcl.exe' failed with exit status 2
*****
Command "d:\python3\python.exe -u -x 'import setuptools; tokenize;__file__='C:\
Users\ADMINI~1\AppData\Local\Temp\pip-build-441rit7\lxml\setup.py';
exec(compile(getattr(tokenize, 'open', open)(__file__).read().replace('\r\n',
'\n'), __file__, 'exec'))" install --record C:\Users\ADMINI~1\AppData\Local\Temp\pip-build-441rit7\lxml\install-record.txt --single-version-externally-managed
--compile" failed with error code 1 in C:\Users\ADMINI~1\AppData\Local\Temp\pip-build-441rit7\lxml\

```

图 11-11 缺少 lxml 出现的错误提示界面

首先，我们可以从 LFD 中下载对应版本的 lxml。

打开以下网址：

<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

然后通过 Ctrl+F 查找 “lxml-”，如图 11-12 所示，找到后我们下载对应版本即可。比如，笔者的电脑是 64 位的，Python 版本是 Python 3.5，所以可以下载 “lxml-3.6.4-cp35-cp35m-win\_amd64.whl”。

下载之后，进入 cmd 命令行。

然后，可以先使用 pip 安装好 wheel：

```
pip install wheel
```

如果未安装 wheel，使用该命令可以直接安装 wheel，如果已安装 wheel，使用该命令就会出现如下图所示信息，不会进行重复安装。

随后，进入下载的 “lxml-3.6.4-cp35-cp35m-win\_amd64.whl” 文件所在的目录，比如，我们将该文件存储在了 “D:/vs2015pro/” 文件夹中，我们可以通过如下图所示方式进入该文件夹。

进入该文件夹后，可以使用 “pip install whl 文件名” 安装对应的 whl 文件。

所以此时，我们可以使用：

```
pip install lxml-3.6.4-cp35-cp35m-win_amd64.whl
```

进行该文件的安装，安装完成之后，会出现相应的 “Successfully” 字样，如图 11-13 所示。

安装完成 lxml 之后，该问题即可解决。

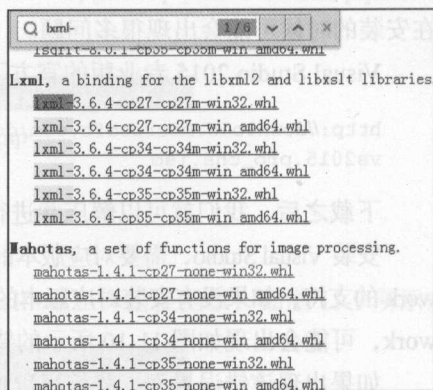


图 11-12 在 LFD 中查找 lxml

```

C:\Users\Administrator.USER-20160828PM>pip install wheel
Requirement already satisfied (use --upgrade to upgrade): wheel in d:\python35\l
ib\site-packages

C:\Users\Administrator.USER-20160828PM>:

D:\>cd vs2015pro

D:\vs2015pro>pip install lxml-3.6.4-cp35-cp35m-win_and64.whl
Processing d:\vs2015pro\lxml-3.6.4-cp35-cp35m-win_and64.whl
Installing collected packages: lxml
Successfully installed lxml-3.6.4

```

图 11-13 cmd 下安装完成 lxml 的提示信息

## 11.2 在 Linux (Centos) 下安装及配置 Scrapy 实战详解

有的朋友使用的操作系统可能是 Linux，那么在 Linux 下如何进行 Scrapy 的安装呢？

同样，我们会以 Python3 为例来讲解如何在 Linux 中安装 Scrapy，我们使用的 Linux 版本是 CentOS7。

在 Linux 中，自带了 Python2.X 的版本，而该自带的 Python2.X 的版本用得比较多，所以我们首先需要在保留现有 Python 版本的基础上，再安装 Python3.X 的版本。

随后，我们可以进行 Scrapy 的安装。

可以总结出在 Linux 中安装 Scrapy 的思路与步骤如下：

1) 在保留 Python2.x 版本的基础上安装 Python3

2) 安装 Scrapy

接下来，我们将分别从安装过程实战和常见问题解决办法两方面来讲解。

### 1. 安装过程实战

(1) 在保留 Python2.x 版本的基础上安装 Python3

首先，在终端中输入 python，可以看到此时 Linux 上已经默认安装了 Python2.7.5 的版本，如下所示：

```

[root@localhost weisuen]# python
Python 2.7.5 (default, Nov 20 2015, 02:00:19)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

```

随后，从 Python 的官网下载 Python3.X 的版本，具体可以从 <https://www.python.org/ftp/python/> 下载，在此我们选择的版本是 Python-3.4.2.tgz，可以按如下代码进行下载：

```
# wget https://www.python.org/ftp/python/3.4.2/Python-3.4.2.tgz
```

下载之后，进行相应的解压操作：

```
# tar -zxvf Python-3.4.2.tgz
```

随后，对 Python3 进行配置：



```
[root@localhost weisuen]# ls
Python-3.4.2 Python-3.4.2.tgz 公共模板视频图片文档下载音乐桌面
[root@localhost weisuen]# cd Python-3.4.2/
[root@localhost Python-3.4.2]# ./configure --prefix=/usr/local/python3
```

配置完成之后, 可以进行 **make** (编译) 和 **make install** (安装):

```
[root@localhost Python-3.4.2]# make
[root@localhost Python-3.4.2]# make install
```

安装完成之后, 为了直接输入 **python** 可以调用刚刚安装的 **Python3**, 需要建立软链接, 在建立软链接之前, 一般需要先备份原来的 **Python**, 具体过程如下:

```
[root@localhost bin]# mv /usr/bin/python /usr/bin/python2bac
[root@localhost bin]# ln -fs /usr/local/python3/bin/python3 /usr/bin/python
```

此时, 输入 **python** 即可调用刚刚安装的 **Python3**, 而输入 **python2.7**, 则可以调用系统原来的 **Python2** 的版本, 两种 **Python** 版本都在 **Linux** 中, 如下所示:

```
[root@localhost bin]# python
Python 3.4.2 (default, Sep 3 2016, 20:04:41)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
[root@localhost bin]# python2.7
Python 2.7.5 (default, Nov 20 2015, 02:00:19)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
```

由于安装 **Scrapy** 时会用到 **pip**, 而在 **Python3.4** 中会默认带有 **pip3**, 所以此时, 为了在终端中输入 **pip3** 可以直接调用 **Python3.4** 自带的 **pip3**, 我们需要为 **pip3** 建立软链接, 如下所示:

```
[root@localhost bin]# ln -fs /usr/local/python3/bin/pip3 /usr/bin/pip3
```

安装好 **pip3** 之后, 在终端中输入 **pip3**, 即可出现如下信息, 说明此时, 在终端中输入 **pip3** 已经能成功调用 **pip3**。

```
[root@localhost bin]# pip3
Usage:
  pip <command> [options]

Commands:
  install      Install packages.
  uninstall    Uninstall packages.
  freeze       Output installed packages in requirements format.
  list         List installed packages.
  show         Show information about installed packages.
```

search	Search PyPI for packages.
wheel	Build wheels from your requirements.
zip	DEPRECATED. Zip individual packages.
unzip	DEPRECATED. Unzip individual packages.
bundle	DEPRECATED. Create pybundles.
help	Show help for commands.

General Options:

-h, --help	Show help.
-v, --verbose	Give more output. Option is additive, and can be used up to 3 times.
-V, --version	Show version and exit.
-q, --quiet	Give less output.
--log-file <path>	Path to a verbose non-appending log, that only logs failures. This log is active by default at /root/.pip/pip.log.
--log <path>	Path to a verbose appending log. This log is inactive by default.
--proxy <proxy>	Specify a proxy in the form [user:passwd@]proxy.server:port.
--timeout <sec>	Set the socket timeout (default 15 seconds).
--exists-action <action>	Default action when a path already exists: (s)witch, (i)gnore, (w)ipe, (b)ackup.
--cert <path>	Path to alternate CA bundle.

## (2) 安装 Scrapy

在完成第一步之后，我们可以通过 pip3 安装 Scrapy。

输入如下指令进行 Scrapy 的安装：

```
[root@localhost ~]# pip3 install scrapy
```

安装完成之后，出现如图 11-14 所示信息。

```
Installing collected packages: scrapy, lxml, service-identity, pyasn1-modules, pyasn1, attrs
  running setup.py install for lxml
    building lxml version 3.6.4.
    building without cython.
    using build configuration of libxslt 1.1.28
    building against libxml2/libxslt in the following directory: /usr/lib64
    building 'lxml.etree' extension
    gcc -pthread -no-unused-result -DDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC
    hon3.4m -c src/lxml/lxml.etree.c -o build/temp.linux-x86_64-3.4/src/lxml/lxml.etree.o -w
    gcc -pthread -shared build/temp.linux-x86_64-3.4/src/lxml/lxml.etree.o -L/usr/lib64 -ls
    thon-34m.so
    building 'lxml.objectify' extension
    gcc -pthread -no-unused-result -DDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC
    hon3.4m -c src/lxml/lxml.objectify.c -o build/temp.linux-x86_64-3.4/src/lxml/lxml.objectify.
    gcc -pthread -shared build/temp.linux-x86_64-3.4/src/lxml/lxml.objectify.o -L/usr/lib64
    tify.cpython-34m.so
Successfully installed scrapy lxml service-identity pyasn1-modules pyasn1 attrs
Cleaning up...
[root@localhost ~]#
```

图 11-14 linux 中 Scrapy 成功安装提示信息

## 2. 常见问题

在 Python 的升级与安装 Scrapy 的过程中，通常会出现各种各样的问题，为了方便读者能够快速解决这些问题，笔者在此总结了本节内容中一些常见的可能会出现的问题与解决办法。

**常见问题 1：配置 Python3 时出现 no acceptable C compiler found in \$PATH**

问题描述：

在编译的时候，可能会出现缺少 C compiler 的情况，具体如下所示：

```
[root@localhost Python-3.4.2]# ./configure --prefix=/usr/local/python3
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for --enable-universalsdk... no
checking for --with-universal-archs... no
checking MACHDEP... linux
checking for --without-gcc... no
checking for gcc... no
checking for cc... no
checking for cl.exe... no
configure: error: in `/home/weisuen/Python-3.4.2':
configure: error: no acceptable C compiler found in $PATH
See `config.log' for more details
```

解决办法：

此时，我们只需要安装对应的 gcc 即可解决问题，具体解决过程如下所示：

```
[root@localhost Python-3.4.2]# yum -y install gcc
```

安装 gcc 之后，即可解决该问题。

**常见问题 2：安装 Python3 时出现 Ignoring ensurepip failure: pip 1.5.6 requires SSL/TLS**

问题描述：

在安装 (make install) Python3 的时候，出现类似如下的错误信息：

```
rm -f /usr/local/python3/share/man/man1/python3.1
(cd /usr/local/python3/share/man/man1; ln -s python3.4.1 python3.1)
if test "xupgrade" != "xno" ; then \
    case upgrade in \
        upgrade) ensurepip="--upgrade" ;; \
        install|*) ensurepip="" ;; \
    esac; \
    ./python -E -m ensurepip \
        $ensurepip --root=/ ; \
fi
Ignoring ensurepip failure: pip 1.5.6 requires SSL/TLS
```

解决办法：

出现这种情况，是因为系统中缺少 openssl-devel，此时我们可以通过 yum 安装 openssl-devel：

```
[root@localhost Python-3.4.2]# yum -y install openssl-devel
```

安装完成之后，该问题即可解决。

**常见问题 3：升级 Python3 后，yum 无法使用**

问题描述：

升级 Python3 后，可能会导致 yum 无法使用，出现如下所示信息：

```
File "/usr/bin/yum", line 30
    except KeyboardInterrupt, e:
        ^
SyntaxError: invalid syntax
```

解决办法：

此时，是因为 /usr/bin/yum 文件中会调用 Python，而此时调用的 Python 为升级后的 Python3.X，由于 Python3.X 与 Python2.X 有一些差异，所以此时，我们可以让系统调用 Python2.X，而此时若要调用原来的 Python2.X 版本，则需要修改一下代码。

编辑文件 /usr/bin/yum：

```
[root@localhost Python-3.4.2]# vim /usr/bin/yum
#!/usr/bin/python
import sys
try:
    import yum
except ImportError:
    print >> sys.stderr, """\n
There was a problem importing one of the Python modules
required to run yum. The error leading to this problem was:

%s

Please install a package which provides this module, or
verify that the module is installed correctly.

It's possible that the above module doesn't match the
current version of Python, which is:

%s

If you cannot solve this problem yourself, please go to
the yum faq at:
http://yum.baseurl.org/wiki/Faq

""" % (sys.exc_value, sys.version)
    sys.exit(1)

sys.path.insert(0, '/usr/share/yum-cli')
try:
    import yummain
    yummain.user_main(sys.argv[1:], exit_code=True)
except KeyboardInterrupt, e:
    print >> sys.stderr, "\n\nExiting on user cancel."
    sys.exit(1)
```

可以发现，此时第一行代码调用的是 python，默认会调用 Python3.X，所以此时我们需要将第一行代码改为：



```
#!/usr/bin/python2.7
```

修改之后，保存并退出。

随后，我们使用 yum 时就不会再出现该问题。

#### 常见问题 4：升级 Python 后 /usr/libexec/urlgrabber-ext-down 出现问题

问题描述：

在升级 Python 后，有时程序在用到 /usr/libexec/urlgrabber-ext-down 文件的时候（比如有时用 yum 之时），可能会出现如下所示的问题。

```
Downloading packages:
Delta RPMs reduced 2.7 M of updates to 731 k (73% saved)
File "/usr/libexec/urlgrabber-ext-down", line 28
    except OSError, e:
        ^
SyntaxError: invalid syntax
File "/usr/libexec/urlgrabber-ext-down", line 28
    except OSError, e:
        ^
SyntaxError: invalid syntax
File "/usr/libexec/urlgrabber-ext-down", line 28
    except OSError, e:
        ^
SyntaxError: invalid syntax
```

由于用户取消而退出

解决办法：

出现这个问题的原因跟问题 3 的原因类似，即程序用到 Python 的时候，无法调用 Python 2.X 的版本去执行。

所以此时，我们可以修改 /usr/libexec/urlgrabber-ext-down 文件里面的代码，具体操作如下所示：

```
[root@localhost Python-3.4.2]# vim /usr/libexec/urlgrabber-ext-down
#!/usr/bin/python
# A very simple external downloader
# Copyright 2011-2012 Zdenek Pavlas

# This library is free software; you can redistribute it and/or
# modify it under the terms of the GNU Lesser General Public
# License as published by the Free Software Foundation; either
# version 2.1 of the License, or (at your option) any later version.
...
```

同样，我们需要将第一行改为：

```
#!/usr/bin/python2.7
```

修改并保存退出之后，该问题即可解决。

**常见问题 5: 安装 Scrapy 的时候出现 bz2 module is not available****问题描述:**

在安装 Scrapy 的时候, 可能会出现类似如下问题:

```
File "/usr/local/python3/lib/python3.4/tarfile.py", line 1566, in open
    return func(name, filemode, fileobj, **kwargs)
File "/usr/local/python3/lib/python3.4/tarfile.py", line 1643, in bz2open
    raise CompressionError("bz2 module is not available")
tarfile.CompressionError: bz2 module is not available
```

Storing debug log for failure in /root/.pip/pip.log

**解决办法:**

出现这个问题, 我们可以安装 Twisted 解决。

首先我们需要下载 Twisted:

```
[root@localhost ~]# wget https://pypi.python.org/packages/source/T/Twisted/Twisted-14.0.0.tar.bz2
```

下载之后, 进行解压和安装, 具体过程如下所示:

```
[root@localhost Twisted-14.0.0]# cd Twisted-14.0.0/
[root@localhost Twisted-14.0.0]# python setup.py install
```

安装完成之后, 会出现如下所示信息:

```
Using /usr/local/python3/lib/python3.4/site-packages
Finished processing dependencies for Twisted==14.0.0
```

此时, 成功解决该问题。

**常见问题 6: 安装 Scrapy 时出现致命错误: libxml/xmlversion.h: 没有那个文件或目录****问题描述:**

在使用 pip3 安装 Scrapy 时, 可能会出现如下错误:

```
gcc -pthread -Wno-unused-result -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes
-fPIC -Isrc/lxml/includes -I/usr/local/python3/include/python3.4m -c src/lxml/
lxml.etree.c -o build/temp.linux-x86_64-3.4/src/lxml/lxml.etree.o -w
```

In file included from src/lxml/lxml.etree.c:321:0:

```
src/lxml/includes/etree_defs.h:14:31: 致命错误: libxml/xmlversion.h: 没有那个文件或目录
#include "libxml/xmlversion.h"
```

编译中断。

Compile failed: command 'gcc' failed with exit status 1

```
creating tmp
cc -I/usr/include/libxml2 -c /tmp/xmlXPathInit8f4a5ufs.c -o tmp/xmlXPathInit8f4a5ufs.o
/tmp/xmlXPathInit8f4a5ufs.c:1:26: 致命错误: libxml/xpath.h: 没有那个文件或目录
#include "libxml/xpath.h"
```

编译中断。

```
error: command 'gcc' failed with exit status 1

*****

Could not find function xmlCheckVersion in library libxml2. Is libxml2 installed?
*****
```

解决办法:

若出现该错误, 我们可以分两步解决: 先安装 libxml2-devel, 再为 libxml 建立软链接, 具体如下所示:

```
[root@localhost ~]# yum -y install libxml2-devel
[root@localhost ~]# ln -fs /usr/include/libxml2/libxml/ /usr/include/libxml
```

### 常见问题 7: 安装 Scrapy 的时候出现 Requirement already satisfied

问题描述:

由于在安装 Scrapy 的过程中会出现一些问题, 所以有的时候我们会重新执行 `pip3 install scrapy`, 但有时 Scrapy 其实已经部分安装, 所以系统有时会出现如下所示信息:

```
[root@localhost ~]# pip3 install scrapy
Requirement already satisfied (use --upgrade to upgrade): scrapy in /usr/local/
python3/lib/python3.4/site-packages
Cleaning up...
```

解决办法:

此时, 无法重新执行 `pip3 install scrapy`, 我们可以先将未完全安装的 Scrapy 拆卸, 再重新执行 `pip3 install scrapy`, 具体过程如下所示:

```
[root@localhost ~]# pip3 uninstall scrapy
[root@localhost ~]# pip3 install scrapy
```

此时该问题即可解决。

**常见问题 8: 安装 Scrapy 时出现: 致命错误: libxslt/xsltconfig.h: 没有那个文件或目录**

问题描述:

在安装 Scrapy 时候, 有时可能会出现如下所示错误提示:

```
gcc -pthread -Wno-unused-result -DDEBBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes
-fPIC -Isrc/lxml/includes -I/usr/local/python3/include/python3.4m -c src/lxml/
lxml.etree.c -o build/temp.linux-x86_64-3.4/src/lxml/lxml.etree.o -w
```

In file included from src/lxml/lxml.etree.c:321:0:

src/lxml/includes/etree\_defs.h:23:32: 致命错误: libxslt/xsltconfig.h: 没有那个文件或目录

```
#include "libxslt/xsltconfig.h"
```

编译中断。

Compile failed: command 'gcc' failed with exit status 1

creating tmp

```
cc -I/usr/include/libxml2 -c /tmp/xmlXPathIniteesnfnu6.c -o tmp/xmlXPathIniteesnfnu6.o
```

```
cc tmp/xmlXPathIniteesnfnu6.o -lxml2 -o a.out
```

error: command 'gcc' failed with exit status 1

解决办法:

此时, 没有 libxslt/xsltconfig.h, 导致该错误的原因是因为缺少 libxslt-devel, 所以此时我们只需要通过 yum 安装 libxslt-devel 即可解决, 具体如下所示:

```
[root@localhost /]# yum -y install libxslt-devel
```

执行完成之后, 该问题即可解决。

**常见问题 9: 安装 Scrapy 后无法使用 Scrapy 指令**

问题描述:

安装好 Scrapy 后, 我们却无法使用 scrapy 指令, 如下所示, 想使用 Scrapy 指令创建一个爬虫项目, 却提示未找到命令。

```
[root@localhost /]# scrapy startproject weisuen
bash: scrapy: 未找到命令 ...
```

解决办法:

遇到这种情况, 我们可以逐步调试解决。



首先，我们无法执行 scrapy 指令，说明此时 /usr/bin/ 目录下没有 scrapy，所以我们需要为 scrapy 建立软链。

首先需要先找出 scrapy 所在地址，如下所示：

```
[root@localhost ~]# find / -name "*scrapy*"
find: '/run/user/1000/gvfs': 权限不够
/usr/local/python3/bin/scrapy
/usr/local/python3/lib/python3.4/site-packages/scrapy
/usr/local/python3/lib/python3.4/site-packages/scrapy/templates/project/scrapy.cfg
```

可以看到，/usr/local/python3/bin/scrapy 路径即是我们要找的路径，此时 scrapy 在 /usr/local/python3/bin/ 目录下，而不在 /usr/bin/ 目录下，故而我们直接输入 scrapy 无法执行对应指令，所以此时为此路径下的 scrapy 建立软链，如下所示：

```
[root@localhost ~]# ln -fs /usr/local/python3/bin/scrapy /usr/bin/scrapy
```

然后就可以执行 scrapy 指令了。

但此时我们执行 scrapy 指令，却会出现新的问题，如下所示：

```
[root@localhost ~]# scrapy
Traceback (most recent call last):
  File "/usr/bin/scrapy", line 7, in <module>
    from scrapy.cmdline import execute
  File "/usr/local/python3/lib/python3.4/site-packages/scrapy/__init__.py", line 27,
  in <module>
    from . import _monkeypatches
  File "/usr/local/python3/lib/python3.4/site-packages/scrapy/_monkeypatches.py", line
  20, in <module>
    import twisted.persisted.styles # NOQA
ImportError: No module named 'twisted.persisted'
```

我们会发现，此时 twisted 模块有问题，而我们之前已经安装过该模块，所以此时可以尝试升级该模块，如下所示：

```
[root@localhost ~]# pip3 install twisted --upgrade
Downloading/unpacking twisted from https://pypi.python.org/packages/6b/23/8dbe86fc8
3215015e221fbd861a545c6ec5c9e9cd7514af114d1f64084ab/Twisted-16.4.1.tar.bz2#md5=c6
d09bdd681f538369659111f079c29d
  Downloading Twisted-16.4.1.tar.bz2 (3.0MB): 3.0MB downloaded
Cleaning up...
Exception:
Traceback (most recent call last):
  File "/usr/local/python3/lib/python3.4/tarfile.py", line 1641, in bz2open
    import bz2
  File "/usr/local/python3/lib/python3.4/bz2.py", line 20, in <module>
    from _bz2 import BZ2Compressor, BZ2Decompressor
ImportError: No module named '_bz2'
```

我们会发现，升级 twisted 的时候显示缺少 \_bz2 模块，此时可以安装 bzip2-devel 以及

bzip2\*, 如下所示:

```
[root@localhost ~]# yum -y install bzip2-devel bzip2*
```

安装好之后, 还需要重新编译与安装 Python, 我们进入 Python3 的安装目录, 然后执行 make 和 make install, 如下所示:

```
[root@localhost Python-3.4.2]# cd /home/weisuen/Python-3.4.2/
[root@localhost Python-3.4.2]# make && make install
```

然后, 再升级 twisted 即可成功升级, 由于过程代码太多, 为了方便读者阅读, “.....”处省略部分执行结果代码, 如下所示:

```
[root@localhost Python-3.4.2]# pip3 install twisted --upgrade
Downloading/unpacking twisted from https://pypi.python.org/packages/6b/23/8dbe86fc
83215015e221fbd861a545c6ec5c9e9cd7514af114d1f64084ab/Twisted-16.4.1.tar.bz2#md5=c
6d09bdd681f538369659111f079c29d
.....
Uninstalling setuptools:
  Successfully uninstalled setuptools
Successfully installed twisted zope.interface setuptools
Cleaning up...
```

随后, 我们可以再次执行 scrapy, 会发现此时缺少 cryptography, 如下所示:

```
[root@localhost Python-3.4.2]# scrapy
.....
from cryptography.hazmat.bindings.openssl.binding import Binding
ImportError: No module named 'cryptography'
```

此时, 我们只需安装 pycrypto、cryptography 即可, 当然在安装之前需要先安装相关的依赖 gcc、libffi-devel、python-devel、openssl-devel, 具体如下所示:

```
[root@localhost Python-3.4.2]# yum -y install gcc libffi-devel python-devel openssl-devel
[root@localhost Python-3.4.2]# yum -y install pycrypto
[root@localhost Python-3.4.2]# pip3 install cryptography
[root@localhost Python-3.4.2]# pip3 install pycrypto
```

安装好后再执行 scrapy 指令即可出现 scrapy 指令相关的提示, 如下所示:

```
[root@localhost Python-3.4.2]# scrapy
Scrapy 1.1.2 - no active project

Usage:
  scrapy <command> [options] [args]
```

Available commands:

bench	Run quick benchmark test
commands	
fetch	Fetch a URL using the Scrapy downloader
genspider	Generate new spider using pre-defined templates

```

runspider      Run a self-contained spider (without creating a project)
settings       Get settings values
shell          Interactive scraping console
startproject    Create new project
version         Print Scrapy version
view           Open URL in browser, as seen by Scrapy

[ more ]      More commands available when run from project directory

```

此时，我们可以尝试用 scrapy 创建一个名为 weisuen 的爬虫项目，如下所示：

```

[root@localhost Python-3.4.2]# scrapy startproject weisuen
New Scrapy project 'weisuen', using template directory '/usr/local/python3/lib/python3.4/
site-packages/scrapy/templates/project', created in:
/home/weisuen/Python-3.4.2/weisuen

```

```

You can start your first spider with:
cd weisuen
scrapy genspider example example.com

```

可以发现，此时爬虫项目 weisuen 成功创建，安装好 Scrapy 后无法使用 scrapy 指令的问题成功解决。

## 11.3 在 MAC 下安装及配置 Scrapy 实战详解

有的朋友使用的是苹果的 MAC 系统，如果此时我们想在 MAC 中使用 Scrapy，首先需要在 MAC 中安装好 Scrapy，同样，我们使用的 Python 版本是 Python3。

MAC 中，同样会自带 Python，目前 MAC 中自带的 Python 版本是 Python2.X，我们可以输入如下代码：

```

weisuendeMini:~ weisuen$ python -V
Python 2.7.11

```

可以看得到，此时系统自带的 Python 版本为：Python 2.7.11。

由于该自带的 Python2.X 版本用的地方比较多，所以此时，我们需要在保留自带 Python 版本的基础上，进行 Python 的升级，换句话说，我们希望 MAC 电脑中同时有 Python2.X 与 Python3.X 的版本。

所以本节内容，将分为两个步骤来讲：

- 1) 在保留 MAC 自带的 Python 版本的基础上对 Python 进行升级。
- 2) 使用 Python3 的 pip 安装 Scrapy。

### 1. Python 版本升级

所以首先，我们进行 1) 保留 MAC 自带的 Python 版本的基础上对 Python 进行升级内容的具体讲解。

进行多版本的 Python 管理，我们可以使用 Homebrew 进行。

首先我们可以去 Homebrew 的官网 ([http://brew.sh/index\\_zh-cn.html](http://brew.sh/index_zh-cn.html)) 下载对应版本的 Homebrew，如图 11-15 所示。



图 11-15 Homebrew 官方页面

此时，我们可以根据其提供的代码 `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"` 进行下载，如下所示：

```
weisuendeMini:~ weisuen$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

下载之后，我们可以通过下载的 brew search 搜索 python 相关的软件，如下所示。

```
weisuendeMini:~ weisuen$ brew search python
app-engine-python      micropython            python3
boost-python           python                  wxpython
gst-python              python-markdown        zpython
```

可以看到，此时出现了很多与 Python 相关的软件，这里有 python，也有 python3。

可以通过 `brew install` 安装搜索出来的 python3，如下所示：

```
weisuendeMini:~ weisuen$ brew install python3
```

安装之后，还需要配置对应的路径信息。

我们首先打开路径配置文件，打开的时候可能需要输入 MAC 的密码，如下所示：

```
weisuendeMini:~ weisuen$ sudo emacs /etc/paths
Password:
```

打开之后，我们如图 11-16 所示进行配置即可：

配置完成之后，我们可以通过 python 调用 MAC 系统自带的 Python2.X 的版本，通过 python3 调用新安

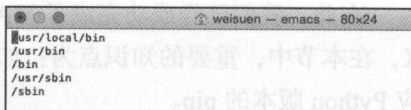


图 11-16 路径配置示例



装的 Python3.X 这个版本，可以通过 `which` 查看，如下所示：

```
weisuendeMini:~ weisuen$ which python
/Library/Frameworks/Python.framework/Versions/2.7/bin/python
weisuendeMini:~ weisuen$ which python3
/Library/Frameworks/Python.framework/Versions/3.4/bin/python3
```

## 2. 安装 Scrapy

那么，对应的 `pip` 版本应该怎么调用呢？

我们可以通过 `pip` 调用 Python2.X 对应的 `pip` 版本，通过 `python3 -m pip` 调用 Python3.X 对应的 `pip` 版本，可以在 `pip` 调用命令后加上 `-V` 参数来查看所调用的 `pip` 版本，如下所示：

```
weisuendeMini:~ weisuen$ pip -V
pip 8.1.2 from /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages (python 2.7)
weisuendeMini:~ weisuen$ python3 -m pip -V
pip 1.5.4 from /Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages (python 3.4)
```

此时，我们可以通过 `python3 -m pip install -U pip` 将 Python3 对应的 `pip` 版本进行升级，如下所示：

```
weisuendeMini:~ weisuen$ python3 -m pip install -U pip
Downloading/unpacking pip from https://pypi.python.org/packages/9c/32/004ce0852e0a127f07f358b715015763273799bd798956fa930814b60f39/pip-8.1.2-py2.py3-none-any.whl#md5=0570520434c5b600d89ec95393b2650b
Downloading pip-8.1.2-py2.py3-none-any.whl (1.2MB): 1.2MB downloaded
Installing collected packages: pip
Found existing installation: pip 1.5.4
Uninstalling pip:
Successfully uninstalled pip
Successfully installed pip
```

随后，我们可以使用 `python3 -m pip install scrapy==1.1.0rc3` 安装对应版本的 `scrapy`，如下所示：

```
weisuendeMini:~ weisuen$ python3 -m pip install scrapy==1.1.0rc3
```

安装好之后，会发现如下所示的安装成功提示信息。

```
Successfully installed PyDispatcher-2.0.5 Twisted-16.4.0 attrs-16.0.0 cffi-1.7.0
cryptography-1.5 cssselect-0.9.2 idna-2.1 lxml-3.6.4 parsel-1.0.3 pyOpenSSL-16.1.0
pyasn1-0.1.9 pyasn1-modules-0.0.8 pycparser-2.14 queuelib-1.4.2 scrapy-1.1.0rc3
service-identity-16.0.0 setuptools-26.0.0 six-1.10.0 w3lib-1.15.0 zope.interface-4.2.0
```

在此，我们已经成功完成了 MAC 下 `Scrapy` 的安装，并且基于的 Python 版本为 Python3.X，在本节中，重要的知识点为：如何进行一个系统下多个 Python 版本的管理、如何使用对应 Python 版本的 `pip`。

## 11.4 小结

本章中，我们分别为大家以不同的操作系统为例讲解了 Scrapy 的安装，在安装的过程中，可能会出现各种各样的问题，这也是本章中的难点。笔者在此总结了常见的问题及对应的解决方案，目的是希望各位读者能够快速解决这些问题，顺利完成 Scrapy 的安装。

接下来我们总结一下本章的一些重点知识：

1) 在 Windows 中安装 Scrapy 的主要思路与步骤为：消除 Fiddler 软件的干扰→安装 Python3→升级 pip→通过 pip 安装 Scrapy。

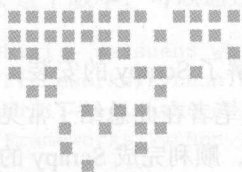
2) 在 Windows 中安装 Scrapy 有时需要 Visual Studio 的支持，建议安装专业版，社区版可能会出现很多问题。笔者使用的版本为 Visual Studio 2015。

3) 在 Linux 中安装 Scrapy 的主要思路与步骤为：保留 Python2.x 版本的基础上安装 Python3→设置好对应的 pip 版本→使用对应版本的 pip 来安装 Scrapy。

4) 在 MAC 中安装 Scrapy 的主要思路与步骤为：保留 Python2.x 版本的基础上安装 Python3→配置好路径→通过对应版本的 pip 安装 Scrapy。



图 11-12-1 Scrapy 安装成功后的终端截图



## Chapter 12 第12章

# 开启 Scrapy 爬虫项目之旅

从本章开始，我们将跟大家一起学习如何使用 Scrapy 框架来开发爬虫项目。使用 Scrapy 框架来开发爬虫项目使用的同样还是 Python 语言，只不过在写法、所用的组件等方面有所不同，我们会为大家系统的介绍如何使用 Scrapy 开发爬虫项目，并主要以实战为导向。

## 12.1 认识 Scrapy 项目的目录结构

在学习如何使用 Scrapy 开发爬虫项目之前，我们首先从总体上认识一下 Scrapy 爬虫项目的目录结构。

使用 Scrapy 创建一个爬虫项目，默认会有如图 12-1 所示的项目结构：

接下来我们分别分析这个项目结构中的各项含义。

首先，会生成一个与爬虫项目名称同名的文件夹，比如此时我们爬虫项目的名称为 weisuen，所以此时，会生成一个名为 weisuen 的文件夹，该文件夹下拥有一个同名子文件夹（可以暂且称为项目核心目录）和一个 scrapy.cfg 文件。

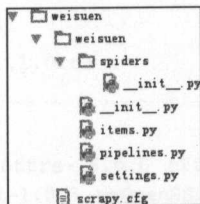


图 12-1 Scrapy 爬虫项目目录结构

该同名子文件夹下放置的是爬虫项目的核心代码，scrapy.cfg 文件主要是爬虫项目的配置文件。

该项目中同名子文件夹 weisuen 下放置了爬虫项目的核心代码，包括一个 spiders 文件夹，以及 \_\_init\_\_.py、items.py、pipelines.py、settings.py 等 Python 文件。

这里提到的 weisuen/weisuen/\_\_init\_\_.py 文件为项目的初始化文件，主要写的是一些项

目的初始化信息。

`weisuen/weisuen/items.py` 文件为爬虫项目的数据容器文件，主要用来定义我们要获取的数据。

`weisuen/weisuen/pipelines.py` 文件为爬虫项目的管道文件，主要用来对 `items` 里面定义的数据进行进一步的加工与处理。

`weisuen/weisuen/settings.py` 文件为爬虫项目的设置文件，主要为爬虫项目的一些设置信息。

`spiders` 文件夹下放置的是爬虫项目中的爬虫部分相关的文件。

`weisuen/weisuen/spiders/__init__.py` 文件为爬虫项目中爬虫部分的初始化文件，主要对 `spiders` 进行初始化。

以上我们对一个基本的 Scrapy 爬虫项目的目录结构进行了分析，在了解了 Scrapy 爬虫项目的基本目录结构之后，我们可以对 Scrapy 爬虫项目有总体的认识，在有了总体的认识之后，可以更方便我们编写爬虫项目。

## 12.2 用 Scrapy 进行爬虫项目管理

接下来，我们讲解如何进行 Scrapy 的爬虫项目管理。

我们需要学习如何使用 Scrapy 创建一个爬虫项目。使用 Scrapy 创建一个爬虫项目，可以使用 `startproject` 命令实现。

首先，在 `cmd` 中进入用来存储新建爬虫项目的文件夹，比如我们要在 “`D:\Python35\myweb\part12\`” 目录中创建 Scrapy 爬虫项目，可以在 `cmd` 中进入该文件夹。

```
C:\>d:
D:\>cd Python35\myweb\part12\
D:\Python35\myweb\part12>
```

此时，可以使用 “`scrapy startproject 项目名`” 来创建一个爬虫项目。

```
D:\Python35\myweb\part12>scrapy startproject myfirstpjt
New Scrapy project 'myfirstpjt', using template directory 'd:\python35\lib\
site-packages\scrapy\templates\project', created in:
D:\Python35\myweb\part12\myfirstpjt
```

```
You can start your first spider with:
cd myfirstpjt
scrapy genspider example example.com
```

在这里，我们创建的爬虫项目名为 `myfirstpjt`，创建成功之后，可以通过 “`cd 爬虫项目所在目录`” 进入该爬虫项目所在的目录。

```
D:\Python35\myweb\part12>cd myfirstpjt
```



```
D:\Python35\myweb\part12\myfirstpjt>
```

进入后，可以对该爬虫项目进行相应管理，此时我们可以通过工具命令实现管理。具体的管理命令在下一节中会详细介绍。

我们打开爬虫项目所在的文件夹，发现此时已经多了一个名为 myfirstpjt 的文件夹，该文件夹就是刚才我们所创建的爬虫项目，如图 12-2 所示。

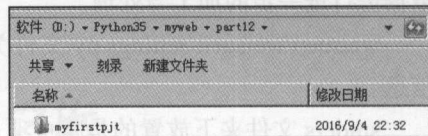


图 12-2 创建的爬虫项目对应的文件夹

使用 scrapy startproject 来创建爬虫项目，在创建的时候我们也可以加上一些参数进行控制，那么有哪些参数呢？我们可以通过 scrapy startproject -h 调出 startproject 的帮助信息，在这里可以看到 scrapy startproject 具体可以添加哪些参数。

```
D:\Python35\myweb\part12>scrapy startproject -h
```

```
Usage
```

```
=====
```

```
scrapy startproject <project_name>
```

```
Create new project
```

```
Options
```

```
=====
```

```
--help, -h show this help message and exit.
```

```
Global Options
```

```
-----
```

```
--logfile=FILE log file. if omitted stderr will be used
```

```
--loglevel=LEVEL, -L LEVEL
```

```
log level (default: DEBUG)
```

```
--nolog disable logging completely
```

```
--profile=FILE write python cProfile stats to FILE
```

```
--pidfile=FILE write process ID to FILE
```

```
--set=NAME=VALUE, -s NAME=VALUE
```

```
set/override setting (may be repeated)
```

```
--pdb enable pdb on failure
```

我们可以对这些重要的参数分别进行分析。

--logfile=FILE 参数主要用来指定日志文件，其中的 FILE 为指定的日志文件的路径地址。

比如，我们希望将日志文件存储在当前目录的上一层目录下，并且日志文件名为 logf.txt，此时，我们可以这样实现：

```
D:\Python35\myweb\part12>scrapy startproject --logfile="../logf.log" mypjt1
```

```
New Scrapy project 'mypjt1', using template directory 'd:\python35\lib\site-packages\scrapy\templates\project', created in:
```

```
D:\Python35\myweb\part12\mypjt1
```

```
You can start your first spider with:
cd mypjtl
scrapy genspider example example.com
```

在当前目录的上一层目录（“D:\Python35\myweb\”），可以看见对应的日志文件 logf.txt，如图 12-3 所示：

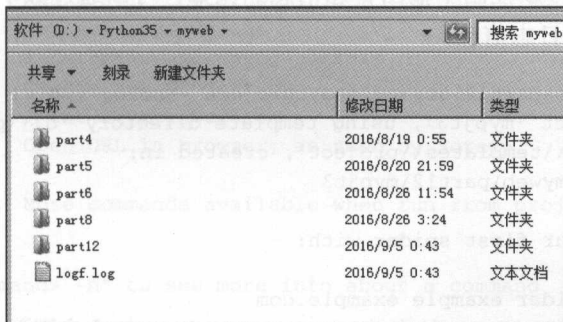


图 12-3 存储的日志文件

该日志文件为刚才创建项目时对应的日志信息，打开如下所示：

```
2016-09-05 00:43:04 [scrapy] INFO: Scrapy 1.1.0rc3 started (bot: scrapybot)
2016-09-05 00:43:04 [scrapy] INFO: Overridden settings: {'LOG_FILE': '../logf.log'}
```

我们发现，此时已经成功通过 --logfile 参数将对应的日志信息写入到了指定的文件中。

--loglevel=LEVEL, -L LEVEL 参数主要用来控制日志信息的等级，默认为 DEBUG 模式，即将对应的调试信息都输出。除了 DEBUG 等级之外，对应的等级还可以设置为其他的值，具体可以设置为的值及对应含义如表 12-1 所示。

表 12-1 日志等级常见值

等级名	含 义
CRITICAL	发生最严重的错误
ERROR	发生了必须立即处理的错误
WARNING	出现一些警告信息，即存在潜在错误
INFO	输出一些提示信息
DEBUG	输出一些调试信息，常用于开发阶段

如下所示，我们将日志等级设置为 DEBUG 最低级别，此时所有调试信息都会输出出来，如果想只输出一些警告以上的日志信息，可以将日志等级设置为 WARNING。

```
D:\Python35\myweb\part12>scrapy startproject --loglevel=DEBUG mypjtl2
2016-09-09 02:15:01 [scrapy] INFO: Scrapy 1.1.0rc3 started (bot: scrapybot)
2016-09-09 02:15:01 [scrapy] INFO: Overridden settings: {}
New Scrapy project 'mypjtl2', using template directory 'd:\python35\lib\site-
```

```
packages\\scrapy\\templates\\project', created in:
D:\\Python35\\myweb\\part12\\mypjt2
```

```
You can start your first spider with:
cd mypjt2
scrapy genspider example example.com
```

通过 `--nolog` 参数可以控制不输出日志信息。比如，可以通过如下程序，实现创建一个爬虫项目，并且不输出日志信息。

```
D:\\Python35\\myweb\\part12>scrapy startproject --nolog mypjt3
New Scrapy project 'mypjt3', using template directory 'd:\\python35\\lib\\site-
packages\\scrapy\\templates\\project', created in:
D:\\Python35\\myweb\\part12\\mypjt3

You can start your first spider with:
cd mypjt3
scrapy genspider example example.com
```

以上，我们分别对 Scrapy 项目创建指令 `startproject` 中的常见重要参数进行了分析，希望各位朋友可以全面的掌握 `startproject` 项目创建指令的使用，可以根据需求创建出自己的爬虫项目。

除了创建爬虫项目之外，如果我们想要删除某个爬虫项目，可以通过直接删除该爬虫项目对应的文件夹来实现。

同时，我们可以使用 Python 编辑器打开爬虫项目中的文件，对对应的文件进行相应的修改，以实现我们的需求。对 Scrapy 爬虫项目中的文件进行编辑，推荐使用 JetBrains PyCharm 编辑器。

## 12.3 常用工具命令

Scrapy 中，工具命令分为两种，一种为全局命令，一种为项目命令。

全局命令不需要依靠 Scrapy 项目就可以在全局中直接运行，而项目命令必须要在 Scrapy 项目中才可以运行。

### 1. 全局命令

首先，我们为大家讲解 Scrapy 全局命令的使用。

那么，在 Scrapy 中有哪些全局命令呢？

要想了解在 Scrapy 中有哪些全局命令，可以在不进入 Scrapy 爬虫项目所在目录的情况下，运行 `scrapy -h`，在 `commands` 下会出现所有的全局命令，如下所示。

```
D:\\Python35\\myweb\\part12>scrapy -h
Scrapy 1.1.0rc3 - no active project
```

Usage:

```
scrapy <command> [options] [args]
```

Available commands:

```
bench          Run quick benchmark test
commands
fetch          Fetch a URL using the Scrapy downloader
runspider      Run a self-contained spider (without creating a project)
settings       Get settings values
shell          Interactive scraping console
startproject   Create new project
version        Print Scrapy version
view           Open URL in browser, as seen by Scrapy
```

```
[ more ]      More commands available when run from project directory
```

Use "scrapy <command> -h" to see more info about a command

可以看到，此时在可用命令（Available commands）中的 commands 栏下，展示出了常见的全局命令，分别为 fetch、runspider、settings、shell、startproject、version、view。

需要注意的是，bench 命令比较特殊，虽在 Available commands 中展现，但仍归为项目命令。

接下来，我们分别为大家讲解这些全局命令的使用。

### （1）fetch 命令

fetch 命令主要用来显示爬虫爬取的过程。

比如，我们可以通过“scrapy fetch 网址”的形式显示出爬取对应网址的过程，从显示爬虫爬取百度首页（http://www.baidu.com）的过程为例，如下所示。

```
D:\Python35\myweb\part12>scrapy fetch http://www.baidu.com
2016-09-10 15:54:31 [scrapy] INFO: Scrapy 1.1.0rc3 started (bot: scrapybot)
2016-09-10 15:54:31 [scrapy] INFO: Overridden settings: {}
2016-09-10 15:54:32 [scrapy] INFO: Enabled extensions:
['scrapy.extensions.logstats.LogStats', 'scrapy.extensions.corestats.CoreStats']
2016-09-10 15:54:33 [scrapy] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',
'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
'scrapy.downloadermiddlewares.retry.RetryMiddleware',
'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',
'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',
'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware',
'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
'scrapy.downloadermiddlewares.chunked.ChunkedTransferMiddleware',
'scrapy.downloadermiddlewares.stats.DownloaderStats']
2016-09-10 15:54:33 [scrapy] INFO: Enabled spider middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
```



```
'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
'scrapy.spidermiddlewares.referer.RefererMiddleware',
'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
'scrapy.spidermiddlewares.depth.DepthMiddleware']
2016-09-10 15:54:33 [scrapy] INFO: Enabled item pipelines:
[]
2016-09-10 15:54:33 [scrapy] INFO: Spider opened
2016-09-10 15:54:33 [scrapy] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items
(at 0 items/min)
2016-09-10 15:54:34 [scrapy] DEBUG: Crawled (200) <GET http://www.baidu.com>
(referer: None)
2016-09-10 15:54:34 [scrapy] INFO: Closing spider (finished)
2016-09-10 15:54:34 [scrapy] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 214,
 'downloader/request_count': 1,
 'downloader/request_method_count/GET': 1,
 'downloader/response_bytes': 1410,
 'downloader/response_count': 1,
 'downloader/response_status_count/200': 1,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2016, 9, 10, 7, 54, 34, 246565),
 'log_count/DEBUG': 1,
 'log_count/INFO': 7,
 'response_received_count': 1,
 'scheduler/dequeued': 1,
 'scheduler/dequeued/memory': 1,
 'scheduler/enqueued': 1,
 'scheduler/enqueued/memory': 1,
 'start_time': datetime.datetime(2016, 9, 10, 7, 54, 33, 674532)}
2016-09-10 15:54:34 [scrapy] INFO: Spider closed (finished)
```

此时，如果在 Scrapy 项目目录之外使用该命令，则会调用 Scrapy 默认的爬虫来进行网页的爬取。如果在 Scrapy 的某个项目目录内使用该命令，则会调用该项目中的爬虫来进行网页的爬取。

我们在使用 fetch 命令的时候，同样可以使用某些参数进行相应的控制。

可以通过 scrapy fetch -h 列出所有可以使用的 fetch 相关参数。

比如，我们可以通过 --headers 参数来控制显示对应的爬虫爬取网页时候的头信息，也可以通过 --nolog 参数来控制不显示日志信息，同时，还可以通过 --spider=SPIDER 参数来控制使用哪个爬虫，通过 --logfile=FILE 参数来指定存储日志信息的文件，通过 --loglevel=LEVEL 参数来控制日志等级。

如下所示，我们分别通过 --headers 参数和 --nolog 参数控制了展现爬虫爬取新浪新闻首页 (<http://news.sina.com.cn/>) 时候的头信息并且不显示日志信息。

```
D:\Python35\myweb\part12>scrapy fetch --headers --nolog http://news.sina.com.cn/
> Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
> User-Agent: Scrapy/1.1.0rc3 (+http://scrapy.org)
```

```
> Accept-Encoding: gzip,deflate
> Accept-Language: en
>
< Content-Type: text/html
< Age: 57
< Cache-Control: max-age=60
< X-Powered-By: shci_v1.03
< Vary: Accept-Encoding
< Server: nginx
< X-Cache: HIT from ja180-189.sina.com.cn
< Date: Sat, 10 Sep 2016 08:07:14 GMT
< Last-Modified: Sat, 10 Sep 2016 08:06:05 GMT
< Expires: Sat, 10 Sep 2016 08:08:14 GMT
```

以上，我们学习了 Scrapy 中 `fetch` 命令的使用，通过 `fetch` 命令我们可以很方便地显示出爬虫爬取某个网页的过程。

## (2) runspider 命令

通过 Scrapy 中的 `runspider` 命令我们可以实现不依托 Scrapy 的爬虫项目，直接运行一个爬虫文件。

下面我们将展示一个运用 `ruspioler` 命令运行爬出文件的例子，首先编写一个 Scrapy 爬虫文件，如下所示。

```
from scrapy.spiders import Spider

class FirstSpider(Spider):
    name = "first"
    allowed_domains = ["baidu.com"]
    start_urls = [
        "http://www.baidu.com",
    ]

    def parse(self, response):
        pass
```

在此，仅需要简单了解该爬虫文件即可，因为在后面我们会详细学习如何编写爬虫文件。首先，定义该爬虫文件的名字为 `first`，同时，定义爬取的起始网址为 `http://www.baidu.com`。

然后，可以使用 `runspider` 命令直接运行该爬虫文件，此时，并不需要依托一个完整的 Scrapy 项目去运行，只需要拥有对应的爬虫文件即可。

如下所示，我们通过 `scrapy runspider` 运行了该爬虫文件，并将日志等级设置成了 `INFO`。

```
D:\Python35\myweb\part12>scrapy runspider --loglevel=INFO first.py
2016-09-10 22:30:46 [scrapy] INFO: Scrapy 1.1.0rc3 started (bot: scrapybot)
2016-09-10 22:30:46 [scrapy] INFO: Overridden settings: {'LOG_LEVEL': 'INFO'}
2016-09-10 22:30:46 [scrapy] INFO: Enabled extensions:
['scrapy.extensions.logstats.LogStats', 'scrapy.extensions.corestats.CoreStats']
```



```
2016-09-10 22:30:46 [scrapy] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.httppauth.HttpAuthMiddleware',
'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
'scrapy.downloadermiddlewares.retry.RetryMiddleware',
'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',
'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',
'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware',
'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
'scrapy.downloadermiddlewares.chunked.ChunkedTransferMiddleware',
'scrapy.downloadermiddlewares.stats.DownloaderStats']
2016-09-10 22:30:46 [scrapy] INFO: Enabled spider middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
'scrapy.spidermiddlewares.referer.RefererMiddleware',
'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
'scrapy.spidermiddlewares.depth.DepthMiddleware']
2016-09-10 22:30:46 [scrapy] INFO: Enabled item pipelines:
[]
2016-09-10 22:30:46 [scrapy] INFO: Spider opened
2016-09-10 22:30:46 [scrapy] INFO: Crawled 0 pages (at 0 pages/min), scraped 0
items (at 0 items/min)
2016-09-10 22:30:47 [scrapy] INFO: Closing spider (finished)
2016-09-10 22:30:47 [scrapy] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 214,
'downloader/request_count': 1,
'downloader/request_method_count/GET': 1,
'downloader/response_bytes': 1410,
'downloader/response_count': 1,
'downloader/response_status_count/200': 1,
'finish_reason': 'finished',
'finish_time': datetime.datetime(2016, 9, 10, 14, 30, 47, 186128),
'log_count/INFO': 7,
'response_received_count': 1,
'scheduler/dequeued': 1,
'scheduler/dequeued/memory': 1,
'scheduler/enqueued': 1,
'scheduler/enqueued/memory': 1,
'start_time': datetime.datetime(2016, 9, 10, 14, 30, 46, 981328)}
2016-09-10 22:30:47 [scrapy] INFO: Spider closed (finished)
```

可以看到，通过该指令在不依靠 Scrapy 项目的情况下最终成功完成了该爬虫文件的运行。

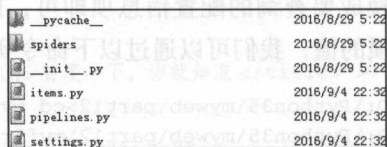
### (3) settings 命令

我们可以通过 Scrapy 中的 settings 命令查看 Scrapy 对应的配置信息。

如果在 Scrapy 项目目录内使用 settings 命令，查看的是对应项目的配置信息，如果在 Scrapy 项目目录外使用 settings 命令，查看的是 Scrapy 默认配置信息。

比如，我们首先进入之前创建的 myfirstpjt 项目中的 myfirstpjt 文件夹（即 “D:\Python35\myweb\part12\myfirstpjt\myfirstpjt”），随后可以看到如图 12-4 所示的文件。

打开其中的 “settings.py” 文件，内容如下所示（篇幅有限，省略了部分信息）：



__pycache__	2016/8/29 5:22
spiders	2016/8/29 5:22
__init__.py	2016/8/29 5:22
items.py	2016/9/4 22:32
pipelines.py	2016/9/4 22:32
settings.py	2016/9/4 22:32

图 12-4 Scrapy 项目核心目录包含的文件图示

```
.....
# http://scrapy.readthedocs.org/en/latest/topics/downloader-middleware.html
# http://scrapy.readthedocs.org/en/latest/topics/spider-middleware.html

BOT_NAME = 'myfirstpjt'

SPIDER_MODULES = ['myfirstpjt.spiders']
NEWSPIDER_MODULE = 'myfirstpjt.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'myfirstpjt (+http://www.yourdomain.com)'

# Obey robots.txt rules
ROBOTSTXT_OBEY = True
.....
```

这些信息就是爬虫项目 myfirstpjt 的配置信息。

我们可以在命令行中进入该项目所在的目录。

```
D:\Python35\myweb\part12>cd myfirstpjt\
```

然后，就可以使用 settings 命令来查看该项目的配置信息。比如，我们可以使用 scrapy settings --get BOT\_NAME 来查看配置信息中 BOT\_NAME 对应的值。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy settings --get BOT_NAME
myfirstpjt
```

可以发现，通过命令查看到的配置信息与对应的 “settings.py” 文件中所写的信息是一致的。所以，如果我们想查看某个爬虫项目的配置信息，可以直接在命令行中进入该项目，使用 scrapy settings 命令查看对应的配置信息项即可。

刚刚我们也提到，如果在 Scrapy 项目目录外使用 settings 命令，查看的是 Scrapy 默认配置信息。比如，从命令行中退出到 Scrapy 项目目录外，然后使用 scrapy settings --get BOT\_NAME 命令查看 Scrapy 默认配置信息中的 BOT\_NAME 信息项，如下所示。

```
D:\Python35\myweb\part12\myfirstpjt>cd ...
D:\Python35\myweb\part12>scrapy settings --get BOT_NAME
scrapybot
```

可以看到，Scrapy 默认的 BOT\_NAME 值是 scrapybot。



如果,要查看配置信息中的其他配置信息项,我们只需要将上述命令中的 BOT\_NAME 位置换成要查询的配置信息项即可,比如要查询 SPIDER\_MODULES (爬虫模块相关) 配置信息项的值,我们可以通过以下命令实现。

```
D:\Python35\myweb\part12>cd myfirstpjt\
D:\Python35\myweb\part12\myfirstpjt>scrapy settings --get SPIDER_MODULES
['myfirstpjt.spiders']
```

此时成功查询到了爬虫项目 myfirstpjt 中的 SPIDER\_MODULES 配置信息项的值。

以上,我们通过 settings 分别查看了 Scrapy 的默认配置信息和 Scrapy 对应项目的配置信息。

#### (4) shell 命令

通过 shell 命令可以启动 Scrapy 的交互终端 (Scrapy shell)。

Scrapy 的交互终端经常在开发以及调试的时候用到,使用 Scrapy 的交互终端可以在不启动 Scrapy 爬虫的情况下,对网站响应进行调试,同样,在该交互终端中,我们也可以写一些 Python 代码进行相应测试。

比如,可以使用 shell 命令,为爬取百度首页创建一个交互终端环境,并设置为不输出日志信息,如下所示:

```
D:\Python35\myweb\part12>scrapy shell http://www.baidu.com --nolog
[s] Available Scrapy objects:
[s]   crawler   <scrapy.crawler.Crawler object at 0x000000000476DBA8>
[s]   item      {}
[s]   request   <GET http://www.baidu.com>
[s]   response  <200 http://www.baidu.com>
[s]   settings  <scrapy.settings.Settings object at 0x0000000004F7A080>
[s]   spider    <DefaultSpider 'default' at 0x512c358>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser
>>>
```

可以看到,在执行了该命令之后,会出现可以使用的 Scrapy 对象及快捷命令,比如 item、response、settings、spider 等,并进入交互模式,在“>>>”后可以输入交互命令及相应的代码。

在该交互模式中,可以提取出爬取到的网页的标题,此时我们通过 XPath 表达式进行提取,可能读者此时并不熟悉 XPath 表达式,我们将会在后面进行 XPath 表达式的基础讲解,在此,仅需要简单知道下面的 XPath 表达式“/html/head/title”的意思是:提取该网页<html>标签下的<head>标签中<title>标签对应的信息。我们知道,此时该标签中的信息即为该网页的标题信息,所以,下面的 XPath 表达式“/html/head/title”目的就是提取爬取到的网页的标题信息。

如下所示,我们通过 sel.xpath 对对应信息进行了提取,并且通过 Python 代码输出了提

取的信息。

```
>>> ti=sel.xpath("/html/head/title")
>>> print(ti)
[<Selector xpath='/html/head/title' data='<title> 百度一下, 你就知道 </title>'>]
>>>
```

可以看到, data 后的内容就是提取到的数据, 成功将百度首页的标题 “<title> 百度一下, 你就知道 </title>” 进行了提取。

除此之外, 在交互终端中我们还可以进行各种开发调试。

如果我们要退出该交互终端, 可以使用 exit() 实现, 如下所示:

```
>>> exit()
D:\Python35\myweb\part12>
```

以上我们分析了 Scrapy 中的 shell 命令如何使用, 学会使用 shell 命令, 在一定程度上可以大大方便爬虫的开发与调试, 因为通过 shell 命令, 我们可以不创建 Scrapy 项目就直接对爬虫进行开发和调试。

### (5) startproject 命令

startproject 命令上一节已经详细分析过, 主要用于创建项目。

### (6) version 命令

通过 version 命令, 可以直接显示 Scrapy 的版本相关信息。

比如, 如果要查看 Scrapy 的版本信息, 可以通过以下代码实现:

```
D:\Python35\myweb\part12>scrapy version
Scrapy 1.1.0rc3
```

如果还想查看与 Scrapy 相关的其他版本信息 (当然包含上述的 Scrapy 版本信息), 比如 twisted/python/platform 的版本的, 可以在 version 命令后加上 -v 参数实现, 如下所示:

```
D:\Python35\myweb\part12>scrapy version -v
Scrapy      : 1.1.0rc3
lxml        : 3.6.4.0
libxml2     : 2.9.4
Twisted     : 16.3.2
Python      : 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)]
pyOpenSSL  : 16.0.0 (OpenSSL 1.0.2h  3 May 2016)
Platform    : Windows-7-6.1.7601-SP1
```

可以看到, 此时已经详细的展示了与 Scrapy 相关的版本信息。

### (7) view 命令

通过 view 命令, 我们可以实现下载某个网页并用浏览器查看的功能。

比如, 我们可以通过如下命令下载网易新闻首页 (<http://news.163.com/>), 并自动用浏览

器查看下载的网页。

```
D:\Python35\myweb\part12>scrapy view http://news.163.com/
```

执行该命令后，会自动打开浏览器并展示已下载到本地的页面（注意观察，由于网页已下载到本地，所以此时的网址是本地的网页地址），如图 12-5 所示。



图 12-5 下载的新闻网页

## 2. 项目命令

以上我们分别分析了 Scrapy 全局命令的使用，那么，除了全局命令之外，Scrapy 项目命令有哪些？具体又应该怎么使用呢？

接下来，我们详细分析 Scrapy 项目命令的使用。

由于 Scrapy 项目命令需要基于 Scrapy 爬虫项目才可以使用，所以我们首先任意进入一个已经创建的 Scrapy 爬虫项目，如下所示。

```
D:\Python35\myweb\part12>cd myfirstpjt\
```

那么，Scrapy 项目命令具体有哪些呢？

同样，可以使用 `scrapy -h` 来查看在项目中可用的命令。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy -h
Scrapy 1.1.0rc3 - project: myfirstpjt
```

Usage:

```
scrapy <command> [options] [args]
```

Available commands:

bench	Run quick benchmark test
check	Check spider contracts
commands	
crawl	Run a spider

edit	Edit spider
fetch	Fetch a URL using the Scrapy downloader
genspider	Generate new spider using pre-defined templates
list	List available spiders
parse	Parse URL (using its spider) and print the results
runspider	Run a self-contained spider (without creating a project)
settings	Get settings values
shell	Interactive scraping console
startproject	Create new project
version	Print Scrapy version
view	Open URL in browser, as seen by Scrapy

Use "scrapy <command> -h" to see more info about a command

在展示出来的命令中，除去上面已经提到的 Scrapy 全局命令，剩下的就是 Scrapy 项目命令。

所以，经过整理，Scrapy 的项目命令主要有：bench、check、crawl、edit、genspider、list、parse。

值得注意的是，Scrapy 全局命令既可以在非 Scrapy 爬虫项目文件夹中使用，同时也可以在本 Scrapy 爬虫项目文件夹中使用，而 Scrapy 项目命令一般只能在 Scrapy 爬虫项目文件夹中使用。

接下来我们将分别讲解 Scrapy 各项目命令的使用方法。

### (1) Bench 命令

使用 bench 命令可以测试本地硬件的性能。

当我们运行 scrapy bench 的时候，会创建一个本地服务器并且会以最大的速度爬行，在此为了测试本地硬件的性能，避免其他过多因素的影响，所以仅进行链接跟进，不进行内容的处理。

如下所示，我们使用了 scrapy bench 对本地硬件的性能进行了测试。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy bench
2016-09-11 14:51:32 [scrapy] INFO: Scrapy 1.1.0rc3 started (bot: myfirstpjt)
2016-09-11 14:51:32 [scrapy] INFO: Overridden settings: {'LOGSTATS_INTERVAL':
1, 'NEWSPIDER_MODULE': 'myfirstpjt.spiders', 'CLOSESPIDER_TIMEOUT': 10, 'LOG_
LEVEL': 'INFO', 'BOT_NAME': 'myfirstpjt', 'ROBOTSTXT_OBEY': True, 'SPIDER_MODULES':
['myfirstpjt.spiders']}
Bench server at http://0.0.0.0:8998
2016-09-11 14:51:34 [scrapy] INFO: Spider opened
2016-09-11 14:51:34 [scrapy] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items
(at 0 items/min)
2016-09-11 14:51:35 [scrapy] INFO: Crawled 62 pages (at 3720 pages/min), scraped 0
items (at 0 items/min)
2016-09-11 14:51:36 [scrapy] INFO: Crawled 118 pages (at 3360 pages/min), scraped 0
items (at 0 items/min)
2016-09-11 14:51:37 [scrapy] INFO: Crawled 166 pages (at 2880 pages/min), scraped 0
items (at 0 items/min)
2016-09-11 14:51:38 [scrapy] INFO: Crawled 214 pages (at 2880 pages/min), scraped 0
```



```

items (at 0 items/min)
2016-09-11 14:51:39 [scrapy] INFO: Crawled 262 pages (at 2880 pages/min),
scraped 0 items (at 0 items/min)
2016-09-11 14:51:40 [scrapy] INFO: Crawled 302 pages (at 2400 pages/min),
scraped 0 items (at 0 items/min)
2016-09-11 14:51:41 [scrapy] INFO: Crawled 350 pages (at 2880 pages/min),
scraped 0 items (at 0 items/min)
2016-09-11 14:51:42 [scrapy] INFO: Crawled 390 pages (at 2400 pages/min),
scraped 0 items (at 0 items/min)
2016-09-11 14:51:43 [scrapy] INFO: Crawled 430 pages (at 2400 pages/min),
scraped 0 items (at 0 items/min)
2016-09-11 14:51:44 [scrapy] INFO: Closing spider (closespider_timeout)
2016-09-11 14:51:44 [scrapy] INFO: Crawled 470 pages (at 2400 pages/min),
scraped 0 items (at 0 items/min)

```

可以看到，在得到的测试结果中，单纯就硬件性能来说，显示每分钟大约能爬 2400 个网页。这只是一个参考标准，在实际运行爬虫项目的时候，会由于各种因素导致速度不同，一般来说，可以根据实际运行的速度与该参考速度进行对比结果，从而对爬虫项目进行优化与改进。

## (2) Genspider 命令

可以使用 `genspider` 命令来创建 Scrapy 爬虫文件，这是一种快速创建爬虫文件的方式。

使用该命令可以基于现有的爬虫模板直接生成一个新的爬虫文件，非常方便。同样，需要在 Scrapy 爬虫项目目录中，才能使用该命令。

可以用该命令的 `-l` 参数来查看当前可以使用的爬虫模板，如下所示。

```

D:\Python35\myweb\part12\myfirstpj>scrapy genspider -l
Available templates:
    basic
    crawl
    csvfeed
    xmlfeed

```

可以看到，当前可以使用的爬虫模板有 `basic`、`crawl`、`csvfeed`、`xmlfeed`。

此时，可以基于其中的任意一个爬虫模板来生成一个爬虫文件，比如，我们可以使用 `basic` 模板生成一个爬虫文件，格式为“`scrapy genspider -t 模板新爬虫名新爬虫爬取的域名`”，如下所示。

```

D:\Python35\myweb\part12\myfirstpj>scrapy genspider -t basic weisuen iqianyue.com
Created spider 'weisuen' using template 'basic' in module:
Myfirstpj.spiders.weisuen

```

执行后，我们成功基于 `basic` 爬虫模板创建了一个新的爬虫文件 `weisuen`，定义爬取的域名为 `iqianyue.com`，在该项目的对应爬虫文件夹下，可以发现新爬虫文件 `weisuen.py`，如图 12-6 所示。

现在我们已经学会如何基于爬虫模板来快速创建新的爬虫文件，假如我们想查看爬虫模板的内容，应该怎么办呢？

可以通过 `-d` 参数实现查看某个爬虫模板的内容，比如我们想查看 `csvfeed` 爬虫模板中的内容，可以通过“`scrapy genspider -d csvfeed`”实现，如下所示。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy genspider -d csvfeed
```

此时，在命令行中便可以查看到对应爬虫模板中的具体内容，比如上面代码的执行结果如图 12-7 所示。

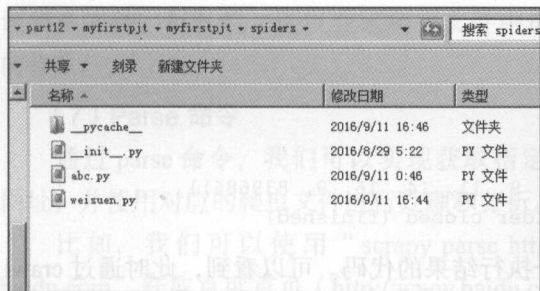


图 12-6 新爬虫文件图示

```
D:\Python35\myweb\part12\myfirstpjt>scrapy genspider -d csvfeed
645 # -*- coding: utf-8 -*-
646
647 from scrapy.spiders import CSVFeedSpider
648
649 from $project_name.items import $(ProjectName)Item
650
651
652 class $classname(CSVFeedSpider):
653     name = '$name'
654     allowed_domains = ['$domain']
655     start_urls = ['http://www.$domain/feed.csv']
656     # headers = {'id', 'name', 'description', 'image_link'}
657     # delimiter = '\t'
658
659     # Do any adaptations you need here
660     #def adapt_response(self, response):
661     #    return response
662
663     def parse_row(self, response, row):
664         i = $(ProjectName)Item()
665         #i['url'] = row['url']
```

图 12-7 查看爬虫模板文件的具体内容

以上我们已经学习了 `genspider` 命令的使用，希望读者可以掌握如何查看当前可使用的爬虫模板、基于某个爬虫模板创建新的爬虫文件、查看爬虫模板中的具体内容等知识。

### (3) Check 命令

爬虫的测试比较麻烦，所以在 Scrapy 中使用合同 (contract<sup>①</sup>) 的方式对爬虫进行测试。

使用 Scrapy 中的 `check` 命令，可以实现对某个爬虫文件进行合同 (contract) 检查。

比如要对刚才基于模板创建的爬虫文件 `weisuen.py` 进行合同 (contract) 检查，我们可以用“`scrapy check 爬虫名`”实现，注意此时“`check`”后面的是爬虫名，不是爬虫文件名，所以是没有后缀的，如下所示。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy check weisuen
```

```
-----
Ran 0 contracts in 0.000s
```

```
OK
```

可以看到，此时该爬虫文件的合同检查通过，显示的结果为“OK”。

### (4) Crawl 命令

可以通过 `crawl` 命令来启动某个爬虫，启动格式是“`scrapy crawl 爬虫名`”。

① 也有翻译为契约，是一种交互式的检查方式。

比如，我们可以启动爬虫项目 `myfirstpjt` 中的 `weisuen` 爬虫，启动命令以及结果如下所示。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy crawl weisuen --loglevel=INFO
2016-09-11 22:16:09 [scrapy] INFO: Scrapy 1.1.0rc3 started (bot: myfirstpjt)
2016-09-11 22:16:09 [scrapy] INFO: Overridden settings: {'NEWSPIDER_MODULE':
'myfirstpjt.spiders', 'SPIDER_MODULES': ['myfirstpjt.spiders'], 'BOT_NAME':
'myfirstpjt', 'ROBOTSTXT_OBEY': True, 'LOG_LEVEL': 'INFO'}
2016-09-11 22:16:09 [scrapy] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats', 'scrapy.extensions.logstats.LogStats']
.....
'finish_time': datetime.datetime(2016, 9, 11, 14, 16, 10, 229687),
'log_count/INFO': 7,
'response_received_count': 2,
'scheduler/dequeued': 2,
'scheduler/dequeued/memory': 2,
'scheduler/enqueued': 2,
'scheduler/enqueued/memory': 2,
'start_time': datetime.datetime(2016, 9, 11, 14, 16, 9, 839686)}
2016-09-11 22:16:10 [scrapy] INFO: Spider closed (finished)
```

由于篇幅有限，在“……”处省略了部分执行结果的代码。可以看到，此时通过 `crawl` 成功启动了 `myfirstpjt` 项目中名为 `weisuen` 的爬虫。

需要注意的是，`crawl` 后面跟的是爬虫名，而不是爬虫项目名。

### (5) List 命令

通过 Scrapy 中的 `list` 命令，可以列出当前可使用的爬虫文件。

比如，我们可以在命令行中进入爬虫项目 `myfirstpjt` 所在的目录，然后使用 `scrapy list` 直接列出当前可以使用的爬虫文件，如下所示。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy list
abc
weisuen
```

可以看到，此时可以使用的爬虫文件有两个，分别是 `abc` 和 `weisuen`。

### (6) Edit 命令

通过 Scrapy 中的 `edit` 命令，我们可以直接打开对应编辑器对爬虫文件进行编辑，该命令在 Windows 中执行会出现一点问题，而且在 Windows 中，我们一般会使用 Python IDE（比如 PyCharm）直接对爬虫项目进行管理和编辑，但是该命令在 Linux 中是很方便的。

比如，要在 Linux 中使用 `edit` 命令编辑某个爬虫文件，我们可以首先使用 `scrapy list` 先列出当前存在的爬虫文件，如下所示。

```
[root@localhost myfirst]# scrapy list
:0: UserWarning: You do not have a working installation of the service_identity
module: 'No module named 'pyasn1_modules''. Please install it from <https://
pypi.python.org/pypi/service_identity> and make sure all of its dependencies are
```

satisfied. Without the service\_identity module and a recent enough pyOpenSSL to support it, Twisted can perform only rudimentary TLS client hostname verification. Many valid certificate/hostname mappings may be rejected.

abc

可以看到, 此时存在爬虫文件 abc, 如果我们要对该爬虫文件进行编辑, 可以直接使用以下命令。

```
[root@localhost myfirst]# scrapy edit abc
```

在执行了该命令后, 会自动出现如图 12-8 界面。

此时, 通过 edit 命令自动调用对应编辑器打开了该爬虫文件, 打开后, 可以在该爬虫文件中进行相应的修改。

### (7) Parse 命令

通过 parse 命令, 我们可以实现获取指定的 URL 网址, 并使用对应的爬虫文件进行处理和分析。

比如, 我们可以使用 “scrapy parse http://www.baidu.com” 获取百度首页 (http://www.baidu.com), 由于在这里没有指定爬虫文件, 也没有指定处理函数, 所以此时会使用默认的爬虫文件和默认的处理函数, 进行相应的处理, 如下所示。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy parse http://www.baidu.com --nolog
```

```
>>> STATUS DEPTH LEVEL 0 <<<
```

```
# Scraped Items -----
[]
# Requests -----
[]
```

“scrapy parse” 命令拥有很多参数, 具体有哪些参数我们可以通过 scrapy parse -h 来看, 如下所示。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy parse -h
```

```
Usage
```

```
=====
```

```
scrapy parse [options] <url>
```

```
Parse URL (using its spider) and print the results
```

```
Options
```

```
=====
```

```
--help, -h
```

```
show this help message and exit
```

```
--spider=SPIDER
```

```
use this spider without looking for one
```

```
-a NAME=VALUE
```

```
set spider argument (may be repeated)
```

```
--pipelines
```

```
process items through pipelines
```

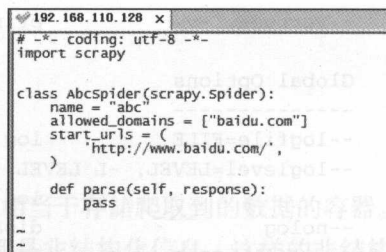


图 12-8 在 Linux 通过 edit 命令直接打开编辑器编辑对应爬虫文件



```
--nolinks          don't show links to follow (extracted requests)
--noitems          don't show scraped items
--nocolour         avoid using pygments to colorize the output
--rules, -r        use CrawlSpider rules to discover the callback
--callback=CALLBACK, -c CALLBACK
                  use this callback for parsing, instead looking for a
                  callback
--depth=DEPTH, -d DEPTH
                  maximum depth for parsing requests [default: 1]
--verbose, -v      print each depth level one by one

Global Options
-----
--logfile=FILE     log file. if omitted stderr will be used
--loglevel=LEVEL, -L LEVEL
                  log level (default: DEBUG)
--nolog            disable logging completely
--profile=FILE     write python cProfile stats to FILE
--pidfile=FILE     write process ID to FILE
--set=NAME=VALUE, -s NAME=VALUE
                  set/override setting (may be repeated)
--pdb              enable pdb on failure
```

可以看到，这些参数大致分为两类：普通参数（Options）和全局参数（Global Options），全局参数我们在其他命令中基本上已经看过，所以在这里我们主要关注该命令对应的参数（Options）。

下面归纳了常用的参数及其含义，如表 12-2 所示。

表 12-2 parse 命令对应的参数表

参 数	含 义
--spider=SPIDER	强行指定某个爬虫文件 spider 进行处理
-a NAME=VALUE	设置 spider 的参数，可能会重复
--pipelines	通过 pipelines 来处理 items
--nolinks	不展示提取到的链接信息
--noitems	不展示得到的 items
--nocolour	输出结果颜色不亮
--rules, -r	使用 CrawlSpider 规则去处理回调函数
--callback=CALLBACK, -c CALLBACK	指定 spider 中用于处理返回的响应的回调函数
--depth=DEPTH, -d DEPTH	设置爬行深度，默认深度为 1
--verbose, -v	显示每层的详细信息

通过上方的表格，可以很清晰地知道 parse 命令中有哪些常见的参数。

比如，如果想指定某个爬虫文件进行处理，可以通过上面的“--spider=SPIDER”参数实现，如下所示。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy parse http://baidu.com --spider=weisuen
--nolog
```

```
>>> STATUS DEPTH LEVEL 0 <<<
```

```
# Scraped Items -----
```

```
[]
```

```
# Requests -----
```

```
[]
```

此时，我们指定了爬虫文件 `weisuen` 进行处理。

## 12.4 实战：Items 的编写

使用 Scrapy 中的 Item 对象可以保存爬取到的数据，相当于存储爬取到的数据的容器。

一般来说，互联网网页中的信息比较庞大，基本上都是非结构化信息，这样的非结构化信息不太利于我们对信息的管理，所以此时，我们可以定义自己所关注的结构化信息，然后从庞大的互联网信息体系中提取出我们关注的结构化信息，这样可以更利于我们对数据的管理，提取之后，这些数据信息需要一个存储的地方，此时，可以将提取到的结构化数据存储到 Item 对象中。

所以，首先需要规划好自己所关注的结构化信息，随后，还需要将这些结构化信息在对应爬虫项目中的 Items 文件中进行相应的定义。

比如，若我们关注的信息项主要有网页标题，网页关键词，网页版权信息，网页地址等，此时，需要在对应爬虫项目中的 Items 文件中对这些规划好的结构化信息进行相应的定义，如果现在使用 `myfirstpjt` 爬虫项目对对应网址进行爬行，就首先需要在 `myfirstpjt` 爬虫项目中的 `items.py` 文件中进行相应的结构化数据的定义。

如图 12-9 所示，首先用编辑器打开 `myfirstpjt` 爬虫项目中的 `items.py` 文件。

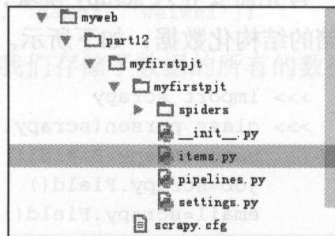


图 12-9 通过编辑器打开 items 文件

打开后，内容如下所示：

```
# -*- coding: utf-8 -*-
# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html
import scrapy
class MyfirstpjtItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
pass
```

在这个文件自动生成的代码中，首先导入了 scrapy，随后定义了一个名为 MyfirstpjtItem 的类，在该类中暂时没有对任何数据进行定义，只有一个 pass 占位语句。

如果我们要对结构化信息进行定义，可以直接修改对应的类，比如此时需要修改的类为 MyfirstpjtItem。

定义结构化数据信息的格式如下：

```
结构化数据名 = scrapy.Field()
```

所以，若要对结构化数据网页标题、网页关键词、网页版权信息、网页地址等进行定义，可以将该类 MyfirstpjtItem 的代码修改为如下：

```
class MyfirstpjtItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    urlname=scrapy.Field()
    urlkey=scrapy.Field()
    urlcr=scrapy.Field()
    urladdr=scrapy.Field()
```

可以看到，要定义一个结构化数据，只需要将 scrapy 下的 Field 类实例化即可。

我们可以通过 Python shell 命令行来实际使用一下 Items，更深入地理解 Items。

首先，需要打开 Python shell（可以直接使用 Python 默认编辑器 IDLE 的 shell 界面），在这个界面中，即可以进行相应 Python 程序的编写。假如此时，我们需要定义一系列结构化数据，可以使用 Items 实现。

首先需要导入 scrapy 模块，然后创建一个继承自 scrapy.Item 的类，在该类中可以定义要存储的结构化数据，如下所示。

```
>>> import scrapy
>>> class person(scrapy.Item):
    name=scrapy.Field()
    job=scrapy.Field()
    email=scrapy.Field()
```

我们定义了一系列结构化数据：人的信息 person，里面分别有 3 项：人名 name，工作 job，邮箱 email。

然后，我们在存储具体数据的时候，只需要实例化该类，如下所示。

```
>>> weisuen=person(name="weiwei",job="teacher",email="qiansyy@iqianyue.com")
```

在此，我们实例化了 person 类，并初始化了参数，初始化格式为：数据项名 = “对应值”。并将实例化后的对象赋值给了变量 weisuen。

我们可以输出该对象，如下所示。

```
>>> print(weisuen)
{'email': 'qiansyy@iqianyue.com', 'job': 'teacher', 'name': 'weiwei'}
```

可以发现，对应的数据会以字典的形式存储，原数据项名会转变为字典中的字段名，原数据项对应的值会转变为字典中相应字段名对应的值，比如原来的 `name="weiwei"` 会转变为字典形式 `"name": "weiwei"`。

如果我们要单独取某个字段所对应的值，可以通过：对象名 [“字段名”] 的方式实现，比如，想单独输出 `weisuen` 对象的 `job` 字段对应的值，可以通过如下方式实现：

```
>>> weisuen["job"]
'teacher'
```

如果我们需要改变某个数据项的值，可以直接通过赋值替换的方法实现，比如需要把邮件改为 `"abc@sina.com"`，可以这样实现：

```
>>> weisuen["email"]
'abc@sina.com'
```

如果想获取该对象中所有的字段名（数据项名），可以通过：对象名 `.keys()` 实现，如下所示：

```
>>> weisuen.keys()
dict_keys(['job', 'email', 'name'])
```

此时成功输出了 `weisuen` 对象下的所有字段名：`job`、`email`、`name`。

如果想获取此时该对象中的项目视图（`ItemsView`），可以通过：对象名 `.items()` 的方式实现，如下所示：

```
>>> weisuen.items()
ItemsView({'email': 'abc@sina.com', 'job': 'teacher', 'name': 'weiwei'})
```

这样就可以将该对象中的 `ItemsView` 输出，在此可以看到我们存储了数据的所有数据项名和对应的数据项值，同样以字典的形式存储。

## 12.5 实战：Spider 的编写

`Spider` 类是 `Scrapy` 中与爬虫相关的一个基类，所有的爬虫文件必须继承该类（`scrapy.Spider`）。

在一个爬虫项目中，爬虫文件是一个极其重要的部分，爬虫所进行的爬取动作以及数据提取等操作都是在该文件中进行定义和编写的，通过爬虫文件，可以定义如何对网站进行相应的爬取。

比如，可以在爬虫项目中通过 `genspider` 命令创建一个爬虫文件，然后对该爬虫文件进行相应的修改与编写。

如图 12-10 所示，我们通过编辑器可以打开之前用

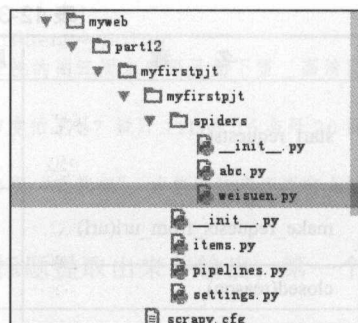


图 12-10 通过编辑器打开爬虫文件 `weisuen.py`



genspider 命令创建的爬虫项目 myfirstpjt 中的爬虫文件 weisuen.py。

打开后, 该文件的默认代码如下所示:

```
# -*- coding: utf-8 -*-
import scrapy
class WeisuenSpider(scrapy.Spider):
    name = "weisuen"
    allowed_domains = ["iqianyue.com"]
    start_urls = (
        'http://www.iqianyue.com/',
    )
    def parse(self, response):
        pass
```

可以看到, 首先在爬虫文件中需要导入 scrapy 模块, 然后创建了一个爬虫类 WeisuenSpider, 该类继承了 scrapy.Spider 基类。

同时, name 属性的值为 "weisuen", name 属性代表的是爬虫名称, 所以此时爬虫名称为 "weisuen"。

allowed\_domains 属性代表的是允许爬行的域名, 如果启动了 OffsiteMiddleware, 非允许的域名对应的网址则会自动过滤掉, 不再跟进。我们设置的允许的域名为 "iqianyue.com"。

start\_urls 属性代表的是爬行的起始网址, 如果没有特别指定爬取的 URL 网址, 则会从该属性中定义的网址开始进行爬行, 在该属性中, 我们可以定义多个起始网址, 网址与网址之间通过逗号隔开。

在这里, 还拥有名为 parse 的方法, 如果没有特别指定回调函数, 该方法是处理 Scrapy 爬虫爬行到的网页响应 (response) 的默认方法, 通过该方法, 可以对响应进行处理并返回处理后的数据, 同时该方法也负责链接的跟进。

除了这些默认生成的属性和方法之外, 在 Scrapy 的 spider 中还有一些常用的属性和方法, 具体如表 12-3 所示。

表 12-3 spider 中其他常用的属性和方法及含义

名 称	属性 or 方法	含 义
start_requests()	方法	该方法会默认读取 start_urls 属性 (当然可以自定义) 中定义的网址, 为每一个网址生成一个 Request 请求对象, 并返回可迭代对象
make_requests_from_url(url)	方法	该方法会被 start_requests() 调用, 该方法负责实现生成 Request 请求对象
closed(reason)	方法	关闭 Spider 时, 该方法会被调用
log(message[, level, component])	方法	使用该方法可以实现 Spider 中添加 log
__init__()	方法	该方法主要负责爬虫的初始化, 为构造函数

比如，我们可以将刚才出现的爬虫文件 `weisuen.py` 进行相应的修改，如下所示：

```
# -*- coding: utf-8 -*-
import scrapy
from myfirstpjt.items import MyfirstpjtItem

class WeisuenSpider(scrapy.Spider):
    name = "weisuen"
    allowed_domains = ["sina.com.cn"]
    start_urls = (
        'http://slide.news.sina.com.cn/s/slide_1_2841_103185.html',
        'http://slide.mil.news.sina.com.cn/k/slide_8_193_45192.html#p=1',
        'http://news.sina.com.cn/pl/2016-09-12/doc-ifxvukhv8147404.shtml',
    )

    def parse(self, response):
        item=MyfirstpjtItem()
        item["urlname"]=response.xpath("/html/head/title/text()")
        print(item["urlname"])
```

在此，除了导入 `scrapy` 模块之外，我们还从 `myfirstpjt.items` 中导入了 `MyfirstpjtItem`，这样就可以使用之前定义的 `Items` 了，导入的格式为“from 项目名 .items import 对应的 `Items` 类名”。

随后，我们在爬虫类中设置了允许的域名为新浪的主域名（`sina.com.cn`），在设置了爬取的起始网址，分别设置了 3 个新浪新闻网页。

接着在 `parse` 方法中，首先实例化了 `MyfirstpjtItem`，并将实例化后的对象赋给 `item` 变量。然后将响应结果中的数据进行提取，我们使用的提取方法是 `XPath` 表达式，“`/html/head/title/text()`”表达式的意思是选择 `<html>` 标签下的 `<head>` 中的 `<title>` 标签，并将 `<title>` 标签中的文本提取出来（详细的 `XPath` 表达式的基础使用在后面会说，在此仅需要知道该表达式的含义即可），并将提取结果赋值给 `item` 对象下的“`urlname`”（`urlname` 在 `items` 文件中已经定义），最后通过 `print()` 输出了提取到的内容。

写好程序之后，可以运行该爬虫，得到对应爬取到的结果，如下所示：

```
D:\Python35\myweb\part12\myfirstpjt>scrapy crawl weisuen --nolog
[<Selector xpath='/html/head/title/text()' data='任性的丽江周末已经开始下雪_高清图集_新浪网'>]
[<Selector xpath='/html/head/title/text()' data='印度怕了吗？疑歼11D歼16与歼20同在高原测试_高清图集_新浪网'>]
[<Selector xpath='/html/head/title/text()' data='告别“政策市”，才能铲除房产谣言土壤|房产|辟谣|政策_新浪新闻'>]
```

可以看到，此时已经成功将这 3 个起始网址中新闻的标题提取出来并输出，第一个 `Scrapy` 项目中自主编写的 `spider` 爬虫文件已经成功运行。

上方的代码中，为什么定义了 `start_urls` 属性就默认定义了起始网址呢？我们是否可以用其他的变量表示起始网址呢？当然是可以的。`start_urls` 属性是默认的设置起始网址属性，如

果我们想用其他的变量(属性)来作为设置起始网址的属性,可以通过重写 `start_requests()` 方法实现。

事实上,如果不重写 `start_requests()` 方法,运行爬虫时会自动的调用该方法生成起始的请求(request),而该方法会默认从 `start_urls` 属性中读取起始网址,所以,若没有重写 `start_requests()` 方法,会默认从 `start_urls` 属性中读取起始网址。

如下所示,将爬虫文件 `weisuen.py` 改写为如下形式:

```
# -*- coding: utf-8 -*-
import scrapy
from myfirstpjt.items import MyfirstpjtItem

class WeisuenSpider(scrapy.Spider):
    name = "weisuen"
    start_urls = (
        'http://slide.news.sina.com.cn/s/slide_1_2841_103185.html',
        'http://slide.mil.news.sina.com.cn/k/slide_8_193_45192.html#p=1',
        'http://news.sina.com.cn/pl/2016-09-12/doc-ixfvukhv8147404.shtml',
    )
    # 定义了新属性 url2
    urls2 = ("http://www.jd.com",
            "http://sina.com.cn",
            "http://yum.iqianyue.com",
    )
    # 重写了 start_requests() 方法
    def start_requests(self):
        # 在该方法中将起始网址设置为从新属性 url2 中读取
        for url in self.urls2:
            # 调用默认 make_requests_from_url() 方法生成具体请求并通过 yield 返回
            yield self.make_requests_from_url(url)
    def parse(self, response):
        item=MyfirstpjtItem()
        item["urlname"]=response.xpath("/html/head/title/text()")
        print(item["urlname"])
```

写好该文件之后,我们可以重新运行该爬虫,结果如下所示:

```
D:\Python35\myweb\part12\myfirstpjt>scrapy crawl weisuen --nolog
[<Selector xpath='/html/head/title/text()' data=' 韬云科技 | 国内首家企业专属云平台 '>]
[<Selector xpath='/html/head/title/text()' data=' 京东 (JD.COM) - 综合网购首选 - 正品低价、品质保障、配送及时、轻松购物! '>]
[<Selector xpath='/html/head/title/text()' data=' 新浪首页 '>]
```

可以看到,此时会爬取我们新属性 `urls2` 中设置的网址的标题,在爬虫文件中,我们通过重写 `start_requests()` 方法,将起始网址设置为了从 `urls2` 属性中读取,并且使用 `for` 循环遍历,每一次循环读取 `urls2` 属性中的一个网址,并调用 Scrapy 中默认的 `make_requests_from_url(url)` 方法来实现生成 Request 请求对象,将生成的请求对象通过 `yield` 返回。

同样,在 `Spider` 类中,其他的方法若有需要我们可以进行重写。

显然，此时我们没有进行链接跟进，关于链接跟进的内容在后面我们会进行深入学习。

## 12.6 XPath 基础

我们知道，之前我们手写网络爬虫的时候，经常使用正则表达式对爬取到的数据进行筛选和提取，而在 Scrapy 中，经常会使用 XPath 表达式进行数据的筛选和提取。

所以在此，我们会对 XPath 表达式进行相应的了解。

XPath 是一种 XML 路径语言，通过该语言可以在 XML 文档中迅速地查找到相应的信息，XPath 表达式通常叫作 XPath selector。

在 XPath 表达式中，使用 “/” 可以选择某个标签，并且可以使用 “//” 进行多层标签的查找。

比如，现在有以下代码：

```
<html>
<head><title> 首页 </title></head>
<body>
<h2>
大数据与爬虫有什么关系？
</h2>
<p> 在大数据处理中，数据源是很重要的，某些时候，数据源无法直接得到，此时使用爬虫可以轻松对大量的数据进行采集，…</p>
<p> 除此之外，它们之间的关系还有…</p>
</body>
</html>
```

如果我们要在以上代码中查询对应信息，可以使用 XPath 表达式进行。

如果要提取出 <h2></h2> 标签对应的内容，可以使用 “/” 选择某个标签，如下所示的 XPath 表达式即可实现。

```
/html/body/h2
```

如果想获取该标签中的文本信息，可以通过 text() 实现，如下所示：

```
/html/body/h2/text()
```

该表达式会提取出如下信息：

“大数据与爬虫有什么关系？”

所以，可以使用 “/” 与 “text()” 对网页代码中的信息进行快速定位。

使用 “//” 可以提取某个标签的所有信息。比如上方代码中出现了多个 <p> 标签，如果想将这所有 <p> 标签的所有信息都提取出来，可以使用 “//” 实现，如下所示：

```
//p
```

在网页代码中，经常还会看到这种形式的代码：



```

<div class="Page Top">
<div class="fl Logo">



<div class="clear"></div>
</div>
.....
</div>

```

如果想获取所有属性 X 的值为 Y 的 <Z> 标签的内容，可以通过 “//Z[@X=” Y”]” 的方式获取。

比如想获取上方代码中所有 class 属性值为 “fl” 的 <img> 标签中的内容，可以通过以下 XPath 表达式获取：

```
//img[@class=" fl" ]
```

以上，是详细的 XPath 表达式使用基础方面的内容，有了这些基础的内容之后，未来我们在写 Scrapy 项目的信息提取的时候，基本上就没问题了。

有了这些基础，如果遇到更复杂的 XPath 表达式，我们也能快速的理解和学习掌握，同时，如果我们需要深入的了解 XPath 表达式，可以直接看 XPath 的官方手册，这一部分深入的知识，并不是本书的内容，读者可以根据爱好自由选择。

## 12.7 Spider 类参数传递

在 Spider 类中，我们还可以通过 -a 选项实现参数的传递。

比如我们希望同一个爬虫程序，在执行的时候可以灵活地传递不同的参数，得到不同的执行结果，此时可以用 Scrapy 参数传递的知识解决。

首先，我们可以在爬虫文件中重写构造方法 \_\_init\_\_(), 在构造方法中设置一个变量用于接收用户在执行该爬虫文件时传递过来的参数，接收到参数后，就可以使用用户传进来的值了，比如将传进来的值设置为起始网址或进行各种处理，这样就可以实现在运行时与爬虫文件内部信息的传递。

然后，在运行时我们只需要通过 -a 选项指定对应的参数名和参数值即可实现参数的传递，非常方便。

比如，可以将爬虫文件 weisuen.py 修改为：

```

# -*- coding: utf-8 -*-
import scrapy
from myfirstpjt.items import MyfirstpjtItem

class WeisuenSpider(scrapy.Spider):
    name = "weisuen"

```

```
# 此时虽然还在此定义了 start_urls 属性，但不起作用，因为在构造方法进行了重写
start_urls = (
    'http://slide.news.sina.com.cn/s/slide_1_2841_103185.html',
    'http://slide.mil.news.sina.com.cn/k/slide_8_193_45192.html#p=1',
    'http://news.sina.com.cn/pl/2016-09-12/doc-ifxvukhv8147404.shtml',
)

# 重写初始化方法 __init__(), 并设置参数 myurl
def __init__(self, myurl=None, *args, **kwargs):
    super(WeisuenSpider, self).__init__(*args, **kwargs)
    # 输出要爬的网址，对应值为接收到的参数
    print("要爬取的网址为: %s" % myurl)
    # 重新定义 start_urls 属性，属性值为传进来的参数值
    self.start_urls = ["%s" % myurl]

def parse(self, response):
    item = MyfirstpjtItem()
    item["urlname"] = response.xpath("/html/head/title/text()")
    print("以下将显示爬取的网址的标题")
    print(item["urlname"])
```

在上方的代码中，我们对构造方法 `__init__()` 进行了重写，并且将 `start_urls` 属性重新赋值为传进来的参数值，这样，在运行该爬虫的时候，就可以传递不同的网址到爬虫文件中，实现同一个爬虫文件爬取不同网址的功能，并且此时会将要爬取的网址的标题输出出来。

然后，可以在 `cmd` 命令行中进入该项目 `myfirstpjt`，然后运行该爬虫文件，并通过 `-a` 选项将 `myurl` 参数的值指定为 `http://www.sina.com.cn`，输出结果如下所示：

```
D:\Python35\myweb\part12\myfirstpjt>scrapy crawl weisuen -a myurl=http://www.sina.com.cn --nolog
要爬取的网址为: http://www.sina.com.cn
以下将显示爬取的网址的标题
[<Selector xpath='/html/head/title/text()' data='新浪首页 '>]
```

可以看到，在运行的时候，通过 `-a` 选项将 `myurl` 参数的值设置为了新浪首页 (`http://www.sina.com.cn`)，执行时成功接收到该参数，并且提取出了传进来的网址的标题信息。

所以，如果我们想要爬取其他网页的标题，可以通过 `-a` 选项指定 `myurl` 参数值为其他网页地址即可，如下所示。我们将使用同一个爬虫文件 `weisuen.py` 爬取其他网址——CSDN 首页 (`http://www.csdn.net`) 的标题。

```
D:\Python35\myweb\part12\myfirstpjt>scrapy crawl weisuen -a myurl=http://www.csdn.net --nolog
要爬取的网址为: http://www.csdn.net
以下将显示爬取的网址的标题
[<Selector xpath='/html/head/title/text()' data='CSDN.NET - 全球最大中文 IT 社区，为 IT 专业技术人员提供最全面的信息传 '>]
```

可以看到，此时对应的信息已经成功爬取并输出。

如果我们要通过传递参数的方式爬取多个网址，应该怎么实现呢？换句话说，我们怎样才能传递多个参数到 Spider 文件呢？

首先我们说一下其中的一种思路，比如可以这么做：我们可以通过 -a 选项在形式上指定一个参数，但是在该参数中包含多个实际参数的信息，但是设置好各实际参数信息之间的间隔符号，随后，在 Spider 文件中，通过 `str.split()` 将各真实参数的信息分割出来，这样就可以使用这多个参数了。

比如，我们要通过参数传递的方式传递多个网址进爬虫文件，可以将爬虫文件 `weisuen.py` 修改为：

```
# -*- coding: utf-8 -*-
import scrapy
from myfirstpjt.items import MyfirstpjtItem

class WeisuenSpider(scrapy.Spider):
    name = "weisuen"

    start_urls = (
        'http://slide.news.sina.com.cn/s/slide_1_2841_103185.html',
        'http://slide.mil.news.sina.com.cn/k/slide_8_193_45192.html#p=1',
        'http://news.sina.com.cn/pl/2016-09-12/doc-ifxvukhv8147404.shtml',
    )

    def __init__(self, myurl=None, *args, **kwargs):
        super(WeisuenSpider, self).__init__(*args, **kwargs)
        # 通过 split() 将传递进来的参数以 "|" 为切割符进行分隔，分隔后生成一个列表并赋值给 myurllist 变量
        myurllist=myurl.split("|")
        # 通过 for 循环遍历该列表 myurllist，并分别输出传进来要爬取的各网址
        for i in myurllist:
            print(" 要爬取的网址为：%s%i")
        # 将起始网址设置为传进来的参数中各网址组成的列表
        self.start_urls=myurllist

    def parse(self, response):
        item=MyfirstpjtItem()
        item["urlname"]=response.xpath("/html/head/title/text()")
        print(" 以下将显示爬取的网址的标题 ")
        print(item["urlname"])
```

在代码中，为了可以使用多个网址，我们需要对传递进来的字符串进行整理，切割为要爬取的各网址，即相当于生成多个实际参数。通过 `split()` 方法处理后，会生成一个列表，列表中的每一个元素就是我们需要的每一个真实参数，随后，即可以处理各个真实参数。在这里，形式上我们仍然以一个参数的形式传递，但实际上我们传递了多个参数的信息。

可以通过下面的代码运行以上程序：

```
D:\Python35\myweb\part12\myfirstpjt>scrapy crawl weisuen -a myurl="http://www.csdn.net|http://yum.iqianyue.com" --nolog
```

要爬取的网址为: `http://www.csdn.net`

要爬取的网址为: `http://yum.iqianyue.com`

以下将显示爬取的网址的标题

```
[<Selector xpath='/html/head/title/text()' data='CSDN.NET - 全球最大中文 IT 社区, 为 IT 专业技术人员提供最全面的信息传 '>]
```

以下将显示爬取的网址的标题

```
[<Selector xpath='/html/head/title/text()' data=' 韬云科技 | 国内首家企业专属云平台 '>]
```

如上所示, 值得注意的是, 此时虽然在形式上我们只传递了一个参数 `myurl`, 但实际上我们传递了两个真实参数信息, 分别为两个网址, 执行后, 成功将这两个网址都进行了爬取并分别提取出了网址的标题。

当然传递多个参数的方法不仅限于此, 在此希望抛砖引玉, 能够让大家掌握多个参数信息传递的手段及方法。

## 12.8 用 XMLFeedSpider 来分析 XML 源

如果想用 Scrapy 爬虫来处理 XML 文件, 我们可以用 XMLFeedSpider 去实现。

我们经常使用 XMLFeedSpider 去处理 RSS 订阅信息。RSS 是一种信息聚合技术, 可以让信息的发布和共享更为高效、便捷。同样, RSS 是基于 XML 标准的。

比如, 如果我们要对新浪博客的 RSS 订阅信息进行分析, 就可以使用 XMLFeedSpider 爬虫实现。

假如现在需要对笔者博客 (`http://blog.sina.com.cn/weiweihappy321`) 的订阅信息进行分析, 我们可以首先打开该博客, 点击页面上的“订阅”按钮, 如图 12-11 所示:

点击了“订阅”按钮之后, 会出现对应的订阅地址, 如图 12-12 所示:

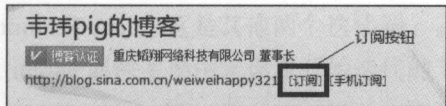


图 12-11 新浪博客中的订阅按钮图示

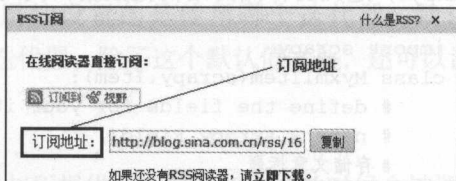


图 12-12 新浪博客对应的订阅地址图示

此时, 我们可以将该订阅地址复制出来, 复制出来后为: `http://blog.sina.com.cn/rss/1615888477.xml`, 可以看得, 该订阅地址为一个 xml 文件, 打开该网址可以出现对应的 XML 格式的内容, 如图 12-13 所示:

接下来我们会创建一个 Scrapy 爬虫项目, 并在项目中创建一个 XMLFeedSpider 爬虫对该 XML 源进行分析及信息的提取。

假如, 我们想提取 XML 文件中的各文章标题、对应链接、作者等信息, 首先, 创建一个 Scrapy 项目。



```
D:\Python35\myweb\part12>scrapy startproject myxml
New Scrapy project 'myxml', using template directory 'd:\\python35\\lib\\site-
packages\\scrapy\\templates\\project', created in:
```

```
D:\Python35\myweb\part12\myxml
```

You can start your first spider with:

```
cd myxml
```

```
scrapy genspider example example.com
```

此时成功创建了一个名为 myxml 的爬虫项目。

随后我们可以编辑该爬虫项目的 items 文件 (D:/Python35/myweb/part12/myxml/myxml

items.py), 定义要存储的结构化数据, 代码如下所示。

```
import scrapy
class MyxmlItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    # 存储文章标题
    title=scrapy.Field()
    # 存储对应链接
    link=scrapy.Field()
    # 存储对应文章作者
    author=scrapy.Field()
```

定义好要存储的结构化数据之后，可以创建一个爬虫文件用于分析 XML 源，如下所示。

```
D:\Python35\myweb\part12>cd myxml\
```

```
D:\Python35\myweb\part12\myxml>scrapy genspider -l
```

Available templates:

basic

crawl

```

csvfeed
xmlfeed
D:\Python35\myweb\part12\myxml>scrapy genspider -t xmlfeed myxmlspider sina.com.cn
Created spider 'myxmlspider' using template 'xmlfeed' in module:
myxml.spiders.myxmlspider

```

在上面的代码中，我们首先进入了该爬虫项目所在的目录，然后使用 `scrapy genspider -l` 查询出当前可使用的爬虫模板文件，因为此时要分析 xml 源，所以爬虫的模板文件选择 `xmlfeed`，随后使用 `scrapy genspider -t` 创建了一个名为 `myxmlspider` 的爬虫文件，允许的域名设置为了 `sina.com.cn`。

爬虫文件创建成功后，可以在项目中的 `spiders` 目录下就可以找到对应的爬虫文件 `myxmlspider.py`，用编辑器打开该爬虫文件，有如下内容：

```

# -*- coding: utf-8 -*-
from scrapy.spiders import XMLFeedSpider
from myxml.items import MyxmlItem

class MyxmlspiderSpider(XMLFeedSpider):
    name = 'myxmlspider'
    allowed_domains = ['sina.com.cn']
    start_urls = ['http://www.sina.com.cn/feed.xml']
    iterator = 'iternodes' # you can change this; see the docs
    itertag = 'item' # change it accordingly

    def parse_node(self, response, selector):
        i = MyxmlItem()
        #i['url'] = selector.select('url').extract()
        #i['name'] = selector.select('name').extract()
        #i['description'] = selector.select('description').extract()
        return i

```

首先我们需要理解一下上述代码，`iterator` 属性设置的是使用哪个迭代器，默认值为 `'iternodes'`，是一个基于正则表达式的高性能的迭代器，除了这个默认值之外，还可以设置为 `'html'` 或 `'xml'`，这是其他两个迭代器。

`itertag` 属性主要用来设置开始迭代的节点。

`parse_node(self, response, selector)` 方法在节点与所提供的标签名相符合的时候会被调用，在该方法中，可以进行一些信息的提取和处理的操作。

除此之外，`XMLFeedSpider` 中还有一些常见的属性与方法，如表 12-4 所示：

表 12-4 XMLFeedSpider 中其他常见属性和方法及含义

名 称	属性 or 方法	含 义
<code>namespaces</code>	属性	以列表的形式存在，主要定义在文档中会被蜘蛛处理的可用命名空间
<code>adapt_response(response)</code>	方法	该方法主要在 spider 分析响应 (Response) 前被调用
<code>process_results(response, results)</code>	方法	该方法主要在 spider 返回结果时被调用，主要对结果在返回前进行最后处理

要提取文章的标题、对应链接、文章作者等信息，我们需要对该 XML 文件进行分析，如图 12-13 所示，如果使用 XPath 表达式进行提取，那么根据观察，文章标题需要使用表达式 `"/rss/channel/item/title/text()"` 进行提取，文章对应链接需要使用表达式 `"/rss/channel/item/link/text()"` 进行提取，文章作者需要使用表达式 `"/rss/channel/item/author/text()"` 进行提取。

接下来我们需要按照需求，对上面的爬虫文件中的代码进行相应的修改，如下所示：

```
# -*- coding: utf-8 -*-
from scrapy.spiders import XMLFeedSpider
from myxml.items import MyxmlItem
class MyxmlspiderSpider(XMLFeedSpider):
    name = 'myxmlspider'
    allowed_domains = ['sina.com.cn']
    # 设置要分析的 XML 文件地址
    start_urls = ['http://blog.sina.com.cn/rss/1615888477.xml']
    iterator = 'iternodes' # you can change this; see the docs
    # 此时将开始迭代的节点设置为第一个节点 rss
    itertag = 'rss' # change it accordingly
    def parse_node(self, response, node):
        i = MyxmlItem()
        # 利用 XPath 表达式将对应信息提取出来，并存储到对应的 Item 中
        i['title'] = node.xpath("/rss/channel/item/title/text()").extract()
        i['link'] = node.xpath("/rss/channel/item/link/text()").extract()
        i['author'] = node.xpath("/rss/channel/item/author/text()").extract()
        # 通过 for 循环以此遍历出提取出来存在 item 中的信息并输出
        for j in range(len(i['title'])):
            print(" 第 "+str(j+1)+" 篇文章 ")
            print(" 标题是: ")
            print(i['title'][j])
            print(" 对应链接是: ")
            print(i['link'][j])
            print(" 对应作者是: ")
            print(i['author'][j])
            print("-----")
        return i
```

然后，可以通过 `scrapy crawl` 运行该爬虫，运行的结果如下所示：

```
D:\Python35\myweb\part12\myxml>scrapy crawl myxmlspider --nolog
```

第 1 篇文章

标题是:

Linux 教程【一】

对应链接是:

[http://blog.sina.com.cn/s/blog\\_6050805d0102vym6.html](http://blog.sina.com.cn/s/blog_6050805d0102vym6.html)

对应作者是:

韦玮 pig

-----

第 2 篇文章

标题是:

襄阳美

对应链接是:

[http://blog.sina.com.cn/s/blog\\_6050805d01016wxn.html](http://blog.sina.com.cn/s/blog_6050805d01016wxn.html)

对应作者是:

韦玮 pig

-----

第 3 篇文章

标题是:

春至华夏

对应链接是:

[http://blog.sina.com.cn/s/blog\\_6050805d01014ak1.html](http://blog.sina.com.cn/s/blog_6050805d01014ak1.html)

对应作者是:

韦玮 pig

-----

# 为了节省资源, 省略部分输出代码.....

-----

第 9 篇文章

标题是:

红梅

对应链接是:

[http://blog.sina.com.cn/s/blog\\_6050805d0100u2o1.html](http://blog.sina.com.cn/s/blog_6050805d0100u2o1.html)

对应作者是:

韦玮 pig

-----

第 10 篇文章

标题是:

残夜

对应链接是:

[http://blog.sina.com.cn/s/blog\\_6050805d0100u1ao.html](http://blog.sina.com.cn/s/blog_6050805d0100u1ao.html)

对应作者是:

韦玮 pig

可以看到, 此时成功爬取了该新浪博客 RSS 订阅文件中的对应信息。

有了这个爬虫之后, 我们同样可以爬取其他新浪博客的订阅信息, 只需要改变一下入口网址即可。

比如, 将入口网址换为其他博客的 RSS 订阅地址: <http://blog.sina.com.cn/rss/1812671331.xml>, 此时, 需要在对应的爬虫文件中修改 `start_urls` 属性, 如下所示:

```
start_urls = ['http://blog.sina.com.cn/rss/1812671331.xml']
```

修改后, 再次运行该爬虫文件, 得到的结果如下所示:

```
D:\Python35\myweb\part12\myxml>scrapy crawl myxmlspider --nolog
```

第 1 篇文章

标题是:

乐视生态终端六连发,「化反」尚需时日

对应链接是:

[http://blog.sina.com.cn/s/blog\\_6c0b2b630102wniv.html](http://blog.sina.com.cn/s/blog_6c0b2b630102wniv.html)

对应作者是:



TECH2IPO 创见

第 2 篇文章

标题是:

降格的美好

对应链接是:

[http://blog.sina.com.cn/s/blog\\_6c0b2b630102wn5j.html](http://blog.sina.com.cn/s/blog_6c0b2b630102wn5j.html)

对应作者是:

TECH2IPO 创见

# 为了节省资源, 省略部分输出代码……

第 9 篇文章

标题是:

在未来, 手机应用将不复存在

对应链接是:

[http://blog.sina.com.cn/s/blog\\_6c0b2b630102wn1d.html](http://blog.sina.com.cn/s/blog_6c0b2b630102wn1d.html)

对应作者是:

TECH2IPO 创见

第 10 篇文章

标题是:

打僵尸也能成为网红? 一大波草根明星正在崛起!

对应链接是:

[http://blog.sina.com.cn/s/blog\\_6c0b2b630102wn16.html](http://blog.sina.com.cn/s/blog_6c0b2b630102wn16.html)

对应作者是:

TECH2IPO 创见

可以看到, 此时我们只换了入口文件, 就可以爬取其他博客的 RSS 订阅信息并可以成功提取标题、对应链接、作者了, 非常方便。

XMLFeedSpider 除了可以爬取 RSS 订阅信息之外, 其他的 XML 文件也能够分析处理, 笔者将一个写好的 XML 文件传到服务器 (文件地址是 <http://yum.iqianyue.com/weisuenbook/pyspd/part12/test.xml>), 供大家测试使用, 对应 XML 文件内容如图 12-14 所示。

假如此时我们想提取出所有人的邮箱 (email), 可以将 XPath 表达式设置为 “/person/email”, 此时可以基于 XMLFeedSpider 创建一个新爬虫文件, 如下所示:

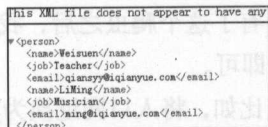


图 12-14 自定义 XML 文件内容

```
D:\Python35\myweb\part12\myxml>scrapy genspider -t xmlfeed person iqianyue.com
Created spider 'person' using template 'xmlfeed' in module:
myxml.spiders.person
```

随后修改该爬虫文件的代码, 如下所示:

```
# -*- coding: utf-8 -*-
from scrapy.spiders import XMLFeedSpider
from myxml.items import MyxmlItem
class PersonSpider(XMLFeedSpider):
    name = 'person'
```

```

allowed_domains = ['iqianyue.com']
# 设置 XML 网址
start_urls = ['http://yum.iqianyue.com/weisuenbook/pyspd/part12/test.xml']
iterator = 'iternodes' # you can change this; see the docs
# 设置开始迭代的节点
itertag = 'person' # change it accordingly

def parse_node(self, response, selector):
    i = MyxmlItem()
    # 提取邮件信息
    i['link'] = selector.xpath('/person/email/text()').extract()
    # 输出提取到的邮件信息
    print(i['link'])
    return i

```

修改好后，可以通过 scrapy crawl 运行该爬虫，运行结果如下所示：

```

D:\Python35\myweb\part12\myxml>scrapy crawl person --nolog
['qiansyy@iqianyue.com', 'ming@iqianyue.com']

```

可以看到，此时已经成功完成了自定义 XML 文件邮件信息的提取，读者在此可以尝试提取其他信息作为练习。

## 12.9 学会使用 CSVFeedSpider

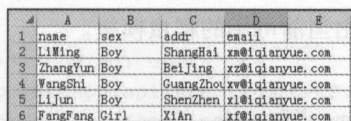
使用爬虫不仅能够处理 XML 文件的数据，还能够处理 CSV 文件的数据。

CSV 文件是一种被用户广泛应用的相对简单、通用的文件格式，其存储的数据可以轻松地与表格数据互相转化。

比如，我们可以新建一个 CSV 文件，然后用表格打开，可以发现数据具有表格形式，随后可以在该表格中写入一些信息，如图 12-15 所示：

这里的第一行是 CSV 文件的列名，第一列存储名字 (name) 信息，第二列存储性别 (sex) 信息，第三列存储地址 (addr) 信息，第四列存储邮箱 (email) 信息，这些数据中每一行表示一个记录。

如果我们将该 CSV 文件以文本文档的方式打开，如图 12-16 所示。



	A	B	C	D	E
1	name	sex	addr	email	
2	LiMing	Boy	ShangHai	xm@iqianyue.com	
3	ZhangYun	Boy	BeiJing	xz@iqianyue.com	
4	WangShi	Boy	GuangZhou	xw@iqianyue.com	
5	LiJun	Boy	ShenZhen	xi@iqianyue.com	
6	FangFang	Girl	XiAn	xf@iqianyue.com	

图 12-15 CSV 文件的表格形式



```

name,sex,addr,email
LiMing,Boy,ShangHai,xm@iqianyue.com
ZhangYun,Boy,BeiJing,xz@iqianyue.com
WangShi,Boy,GuangZhou,xw@iqianyue.com
LiJun,Boy,ShenZhen,xi@iqianyue.com
FangFang,Girl,XiAn,xf@iqianyue.com

```

图 12-16 CSV 文件的文本文档形式

我们可以对比一下文本文档中的数据和表格中的数据，发现其中的对应关系。上面这两张图表现的是同一个文件 mydata.csv，只是不同的展现形式而已，既可以以文本文档的方式

展现,也可以以表格的方式展现。

可以发现,文本文档中的数据与表格中的数据是能够对应起来的,首先,表格中的每行数据对应文本文档中的每行数据,表格中的每列数据与文本文档中的每列数据亦能对应,但是,文本文档中的每列数据通过“,”划分。

CSV 文件最原始的形式是纯文本的形式,由于其具有特定规律,比如列之间通过“,”间隔,行之间通过换行间隔,所以为了方便编辑 CSV 文件,可以根据这些规律轻松地将这些文本文档格式的数据存储转化为表格形式的存储。

那么,如何使用爬虫去处理这些数据呢?

为了方便读者进行测试,笔者首先将该 CSV 文件传到网络服务器中,传后地址为:  
<http://yum.iqianyue.com/weisuenbook/pyspd/part12/mydata.csv>

然后,我们在本地创建一个 Scrapy 爬虫项目,如下所示。

```
D:\Python35\myweb\part12>scrapy startproject mycsv
New Scrapy project 'myscv', using template directory 'd:\python35\lib\site-packages\scrapy\templates\project', created in:
D:\Python35\myweb\part12\myscv
```

```
You can start your first spider with:
cd mycsv
scrapy genspider example example.com
```

假如要用爬虫提取该文件中的所有人的姓名、性别等信息,我们可以编写该爬虫项目下的 Items 文件,定义好关注的数据,如下所示:

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
import scrapy
class MyscvItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    name=scrapy.Field()
    sex=scrapy.Field()
```

我们定义了 **name** 用来存储名字信息,定义了 **sex** 用来存储性别信息。

接着我们在 CMD 命令行中进入该爬虫项目,并列出现当前可用的爬虫模板。

```
D:\Python35\myweb\part12>cd mycsv
D:\Python35\myweb\part12\myscv>scrapy genspider -l
Available templates:
basic
crawl
csvfeed
xmlfeed
```

可以看到,此时有一个爬虫模板为 csvfeed,我们要创建一个爬虫文件处理 CSV 文件,可以依据该爬虫模板创建一个爬虫文件,如下所示。

```
D:\Python35\myweb\part12\mycsv>scrapy genspider -t csvfeed mycsvspider iqianyue.com
Created spider 'mycsvspider' using template 'csvfeed' in module:
mycsv.spiders.mycsvspider
```

随后我们可以编辑创建的爬虫文件 mycsvspider.py(D:\Python35\myweb\part12\mycsv\mycsv\spiders\mycsvspider.py),该文件默认代码如下所示:

```
# -*- coding: utf-8 -*-
from scrapy.spiders import CSVFeedSpider
from mycsv.items import MycsvItem

class MycsvspiderSpider(CSVFeedSpider):
    name = 'mycsvspider'
    allowed_domains = ['iqianyue.com']
    start_urls = ['http://www.iqianyue.com/feed.csv']
    # headers = ['id', 'name', 'description', 'image_link']
    # delimiter = '\t'

    # Do any adaptations you need here
    #def adapt_response(self, response):
    #    return response

    def parse_row(self, response, row):
        i = MycsvItem()
        #i['url'] = row['url']
        #i['name'] = row['name']
        #i['description'] = row['description']
        return i
```

我们需要对这些代码中新知识部分进行初步理解。

代码中 headers 属性主要存放在 CSV 文件中包含的用于提取字段的行信息的列表。delimiter 属性主要存放字段之间的间隔符,parse\_row() 方法主要用来接收一个 response 对象,并进行对应处理。

所以,如果我们要提取该文件中所有人的姓名、性别等信息,可以将 Spider 文件修改为:

```
# -*- coding: utf-8 -*-
from scrapy.spiders import CSVFeedSpider
from mycsv.items import MycsvItem

class MycsvspiderSpider(CSVFeedSpider):
    name = 'mycsvspider'
    allowed_domains = ['iqianyue.com']
    # 定义要处理的 csv 文件所在的网址
    start_urls = ['http://yum.iqianyue.com/weisuenbook/pyspd/part12/mydata.csv']
```



```

# 定义 headers
headers = ['name', 'sex', 'addr', 'email']
# 定义间隔符
delimiter = ','
# Do any adaptations you need here
def adapt_response(self, response):
    # return response
def parse_row(self, response, row):
    i = MycsvItem()
    # 提取各行的 name 这一列的信息
    i['name'] = row['name'].encode()
    # 提取各行的 sex 这一列的信息
    i['sex'] = row['sex'].encode()
    # 进行信息输出
    print("名字是:")
    print(i['name'])
    print("性别是:")
    print(i['sex'])
    # 输出完一个记录的对应列的信息后, 输出一个间隔符, 显示起来方便观察
    print("-----")
    return i

```

然后运行该爬虫, 得到的结果如下所示:

```
D:\Python35\myweb\part12\mycsv>scrapy crawl mycsvspider --nolog
```

```
名字是:
```

```
b'name'
```

```
性别是:
```

```
b'sex'
```

```
...为了节省篇幅资源, 更方便读者阅读, 此处省略部分输出代码...
```

```
名字是:
```

```
b'LiJun'
```

```
性别是:
```

```
b'Boy'
```

```
名字是:
```

```
b'FangFang'
```

```
性别是:
```

```
b'Girl'
```

可以看到, 此时已经将对应 CSV 文件中的各记录的姓名和性别等信息提取并输出了出来。

## 12.10 Scrapy 爬虫多开技能

我们知道, 现在运行 Scrapy 项目中的爬虫文件, 需要一个一个地运行, 那么是否可以将

对应的想运行的爬虫文件批量运行呢？如果可以，又该怎么实现呢？

在 Scrapy 中，如果想批量运行爬虫文件，常见的有两种方法：

- 1) 使用 `CrawProcess` 实现
- 2) 使用修改 `crawl` 源码 + 自定义命令的方式实现

第 1 种方法在 Scrapy 官方文档中已经详细讲到（地址是：<http://doc.scrapy.org/en/latest/topics/practices.html>），所以在此就不过多阐述，如果英文不太好可以在网上找翻译版。

接下来我们将重点通过实战讲解一下第二种方法。

第 2 种方法也是我们使用的比较多的方法，同时学会后使用起来也会比较方便，当然学习的过程中可能比第 1 种方法略难。但相信经过笔者的分析与读者的努力，也能够很容易掌握这种方法。

首先，创建一个 Scrapy 爬虫项目，如下所示：

```
D:\Python35\myweb\part12>scrapy startproject mymultispd
New Scrapy project 'mymultispd', using template directory 'd:\python35\lib\site-packages\scrapy\templates\project', created in:
D:\Python35\myweb\part12\mymultispd
```

You can start your first spider with:

```
cd mymultispd
scrapy genspider example example.com
```

然后，进入该爬虫项目所在目录，并在该项目中创建 3 个爬虫文件（目的是为了创建多个爬虫文件，以供待会运行），如下所示：

```
D:\Python35\myweb\part12>cd mymultispd

D:\Python35\myweb\part12\mymultispd>scrapy genspider -t basic myspd1 sina.com.cn
Created spider 'myspd1' using template 'basic' in module:
mymultispd.spiders.myspd1

D:\Python35\myweb\part12\mymultispd>scrapy genspider -t basic myspd2 sina.com.cn
Created spider 'myspd2' using template 'basic' in module:
mymultispd.spiders.myspd2

D:\Python35\myweb\part12\mymultispd>scrapy genspider -t basic myspd3 sina.com.cn
Created spider 'myspd3' using template 'basic' in module:
mymultispd.spiders.myspd3
```

此时，我们已经在项目中创建了 3 个爬虫文件，有了这些准备工作之后，我们可以正式进入运行多个爬虫文件的功能的编写。

我们知道，在 Scrapy 中，运行一个爬虫文件要通过 `crawl` 命令进行。

Scrapy 是开源的，如果要想实现运行多个爬虫文件，我们参考 `crawl` 命令的源码，进行相应的修改，是否就能够实现呢？

理论上是可行的，经过验证，实际情况也是可行的。

我们可以根据 Scrapy 中 crawl 命令的源码，进行相应的修改，并写一个自己的 Python 文件，这相当于定义了一个新命令，所以还需要使用 Scrapy 添加自定义命令的功能为我们所写的代码添加一个自定义命令。然后就可以根据这个自定义命令，运行多个爬虫文件。

接下来我们进行实战讲解。

要修改 crawl 命令的源码，首先我们需要大概了解 crawl 的源码及含义。

crawl 命令的源码可以在 scrapy 官方的 github 项目中找到（地址是：<https://github.com/scrapy/scrapy/blob/master/scrapy/commands/crawl.py>），找到并打开该源码文件，如下所示，关键信息已经给出注释：

```
import os
from scrapy.commands import ScrapyCommand
from scrapy.utils.conf import arglist_to_dict
from scrapy.utils.python import without_none_values
from scrapy.exceptions import UsageError

# 建立一个命令类继承自 ScrapyCommand
class Command(ScrapyCommand):

    requires_project = True

    def syntax(self):
        return "[options] <spider>"

    def short_desc(self):
        return "Run a spider"

    def add_options(self, parser):
        ScrapyCommand.add_options(self, parser)
        parser.add_option("-a", dest="spargs", action="append", default=[], metavar="NAME=VALUE",
                           help="set spider argument (may be repeated)")
        parser.add_option("-o", "--output", metavar="FILE",
                           help="dump scraped items into FILE (use - for stdout)")
        parser.add_option("-t", "--output-format", metavar="FORMAT",
                           help="format to use for dumping items with -o")

    def process_options(self, args, opts):
        ScrapyCommand.process_options(self, args, opts)
        try:
            opts.spargs = arglist_to_dict(opts.spargs)
        except ValueError:
            raise UsageError("Invalid -a value, use -a NAME=VALUE", print_help=False)
        if opts.output:
            if opts.output == '-':
                self.settings.set('FEED_URI', 'stdout:', priority='cmdline')
            else:
                self.settings.set('FEED_URI', opts.output, priority='cmdline')
```

```

feed_exporters = without_none_values(
    self.settings.getwithbase('FEED_EXPORTERS'))
valid_output_formats = feed_exporters.keys()
if not opts.output_format:
    opts.output_format = os.path.splitext(opts.output)[1].replace(".", "")
if opts.output_format not in valid_output_formats:
    raise UsageError("Unrecognized output format '%s', set one"
        " using the '-t' switch or as a file extension"
        " from the supported list %s" % (opts.output_format,
            tuple(valid_output_formats)))
self.settings.set('FEED_FORMAT', opts.output_format, priority='cmdline')
#run() 方法, 在此可以指定运行哪个爬虫, 要运行所有爬虫, 关键修改该方法
def run(self, args, opts):
    if len(args) < 1:
        raise UsageError()
    elif len(args) > 1:
        raise UsageError("running 'scrapy crawl' with more than one spider is
            no longer supported")
    spname = args[0]

    self.crawler_process.crawl(spname, **opts.spargs)
    self.crawler_process.start()

```

在上面 `crawl` 命令的源码文件中, `Command` 类下面的 `run()` 方法中指定了要运行哪些爬虫文件, 具体通过 `crawler_process.crawl(spname, **opts.spargs)` 实现爬虫文件的运行, `spname` 指的是爬虫名。所以, 我们要实现一次运行多个爬虫文件, 关键需要修改 `run()` 方法, 同时, 我们还需要解决一个问题, 就是如何获取所有的爬虫文件, 如果要获取所有的爬虫文件, 可以通过 `crawler_process.spider_loader.list()` 实现。

所以, 有了上面的基础与思路之后, 我们就可以修改 `crawl` 命令的源码来建立自己的自定义命令所对应的 Python 源码。

首先, 建立一个文件夹存放要写的源码文件, 文件夹名称可以自定义, 满足文件夹命名规则即可, 位置放置在 `spiders` 目录的同级目录下 (在本项目中, 新文件夹位置在 “`D:/Python35/myweb/part12/mymultispd/mymultispd/`” 下)。

在这里, 我们将新文件夹的名字命名为 `mycmd`, 在对应目录下创建该文件夹, 如下所示:

```

D:\Python35\myweb\part12\mymultispd>cd .\mymultispd\
D:\Python35\myweb\part12\mymultispd\mymultispd>mkdir mycmd

```

然后, 进入新建的文件夹并创建一个 Python 文件 (文件名可以自定义, 满足 Python 文件命名规则即可), 如下所示:

```

D:\Python35\myweb\part12\mymultispd\mymultispd>cd .\mycmd\
D:\Python35\myweb\part12\mymultispd\mymultispd\mycmd>echo #>mycrawl.py
#

```



码，在新建的 mycrawl.py 文件中写一个可以一次运行多个爬虫文件的命令的源码了。

首先，将 crawl 命令的源码复制到该文件（mycrawl.py）中，然后进行如下修改：

[illegible]

```

        self.settings.set('FEED_FORMAT', opts.output_format, priority='cmdline')
# 主要修改这里
def run(self, args, opts):
    # 获取爬虫列表
    spd_loader_list=self.crawler_process.spider_loader.list()
    # 遍历各爬虫
    for spname in spd_loader_list or args:
        self.crawler_process.crawl(spname, **opts.spargs)
        print(" 此时启动的爬虫为: "+spname)
    self.crawler_process.start()

```

可以看到, 此时我们主要对 `run()` 方法进行了修改, 在 `run()` 方法中, 先获取的爬虫列表, 然后再依次通过 `for` 循环遍历各爬虫, 遍历时使用 `crawler_process.crawl()` 运行当前得到的 `spider` 爬虫文件, 并输出当前爬虫文件的信息, 便于调试与观察。

然后在该文件的同级目录下 (本项目中, 即 “D:\Python35\myweb\part12\mymultispd\my multispd\mycmd\” 目录) 添加一个初始化文件 `__init__.py`, 如下所示:

```

D:\Python35\myweb\part12\mymultispd\mymultispd\mycmd>echo #>__init__.py
#

```

随后我们需要添加一个自定义命令。

可以在项目配置文件 (`settings.py`) 中进行相应配置, 格式为 “`COMMANDS_MODULE = '项目核心目录.自定义命令源码目录'`”, 具体如下所示:

```

COMMANDS_MODULE = 'mymultispd.mycmd'

```

随后, 在命令行中进入该项目所在目录, 并输入 `scrapy -h`, 出现如下所示信息:

```

D:\Python35\myweb\part12\mymultispd\mymultispd>scrapy -h
Scrapy 1.1.0rc3 - project: mymultispd

```

Usage:

```

scrapy <command> [options] [args]

```

Available commands:

bench	Run quick benchmark test
check	Check spider contracts
commands	
crawl	Run a spider
edit	Edit spider
fetch	Fetch a URL using the Scrapy downloader
genspider	Generate new spider using pre-defined templates
list	List available spiders
mycrawl	Run all spider
parse	Parse URL (using its spider) and print the results
runspider	Run a self-contained spider (without creating a project)
settings	Get settings values

```

shell      Interactive scraping console
startproject Create new project
version    Print Scrapy version
view       Open URL in browser, as seen by Scrapy

```

Use "scrapy <command> -h" to see more info about a command

可以看到，此时已经出现自定义命令 **mycrawl** 了。

随后我们就可以使用该自定义命令同时启动所有爬虫文件了，如下所示：

```
D:\Python35\myweb\part12\mymultispd\mymultispd>scrapy mycrawl --nolog
```

此时启动的爬虫为：myspd1

此时启动的爬虫为：myspd3

此时启动的爬虫为：myspd2

此时已经通过自定义的 **mycrawl** 命令同时启动了爬虫文件 **myspd1**、**myspd2** 和 **myspd3**。

## 12.11 避免被禁止

所谓 **ban**，即禁止的意思。

我们在运行爬虫的时候，如果爬取的网页较多，经常会遇到这种问题。因为现在很多网站都有相应的反爬虫机制，避免爬虫的恶意爬取。

所以，当我们要爬取大量网页的时候，很可能会受到对方服务器的限制，从而被禁止，显然，这不是我们想要的结果。

我们在之前纯手写爬虫项目的时候已经介绍过相应的反爬虫机制以及应对策略，那么在 Scrapy 爬虫项目中，我们应该如何应对这些反爬虫机制呢？

在 Scrapy 项目中，主要可以通过以下方法来避免被禁止：

- 1) 禁止 Cookie；
- 2) 设置下载延时；
- 3) 使用 IP 池；
- 4) 使用用户代理池；
- 5) 其他方法，比如进行分布式爬取等。

接下来我们分别为大家讲解这些避免被 **ban** 的方法。

### 1. 禁止 Cookie

有的网站会通过用户的 Cookie 信息对用户进行识别和分析，此时我们可以通过禁用本地 Cookies 信息让对方网站无法识别出我们的会话信息，从而无法禁止我们的爬取。

如果我们要禁止使用 Cookie，可以在对应的 Scrapy 爬虫项目中的 **settings.py** 文件中进行相应的设置。

打开对应的 Scrapy 爬虫项目中的 settings.py 文件，可以发现文件中有以下两行代码：

```
# Disable cookies (enabled by default)
#COOKIES_ENABLED = False
```

这两行代码都是被注释的状态，这两行代码就是设置禁止使用 Cookie 的代码，我们若要禁用 Cookie，只需要把 #COOKIES\_ENABLED = False 的注释去掉即可，修改为如下代码：

```
# Disable cookies (enabled by default)
COOKIES_ENABLED = False
```

这样，就可以禁用本地的 Cookie，让那些通过用户的 Cookie 信息对用户进行识别和分析的网站无法识别我们，即无法禁止我们爬取。

## 2. 设置下载延时

有的网站会通过我们对网页的访问（爬取）频率进行分析，如果爬取频率过快，则判断为爬虫自动爬取行为，识别后对我们进行相应限制，比如禁制我们爬取该服务器上的网页等。

对于这一类网站，我们只需要控制一下爬行时间间隔即可。

在 Scrapy 爬虫项目中，我们可以直接在对应的 Scrapy 爬虫项目中的 settings.py 文件中进行相应的设置即可。

打开 settings.py 文件，会发现如下几行代码：

```
# Configure a delay for requests for the same website (default: 0)
# See http://scrapy.readthedocs.org/en/latest/topics/settings.html#download-delay
# See also autothrottle settings and docs
#DOWNLOAD_DELAY = 3
```

如果要设置网页下载时间间隔，通过这几行代码配置即可。

上面代码中的前 3 行为配置说明信息，第 4 行 #DOWNLOAD\_DELAY = 3 为设置时间间隔的具体代码，解除这一行的注释即可实现下载延时的配置，其中 3 代表 3 秒，如果想将爬虫下载网页的时间间隔设置为 0.5 秒，将对应的值改为 0.5 即可。

比如，我们想将爬虫下载网页的时间间隔设置为 0.7 秒，可以将上述代码改为：

```
# Configure a delay for requests for the same website (default: 0)
# See http://scrapy.readthedocs.org/en/latest/topics/settings.html#download-delay
# See also autothrottle settings and docs
DOWNLOAD_DELAY = 0.7
```

设置好之后，就可以避免被这一类反爬虫机制的网站禁止。

## 3. 使用 IP 池

有的网站会对用户的 IP 进行检测，如果同一个 IP 在短时间内对自己服务器上的网页进



行大量的爬取，那么可以初步判定为网络爬虫的自动爬取行为，如有必要，该网站可以对该 IP 进行封禁。

作为爬虫方，如果我们的 IP 被封禁了，就需要更换 IP，对于普通用户来说，IP 资源可能会有有限，那么怎么样才能有较多的 IP 呢？

我们之前提到过代理服务器，利用不同的代理服务器我们可以获得不同的 IP，所以此时我们可以获取多个代理服务器，将这些代理服务器的 IP 组成一个 IP 池，爬虫每次对网页进行爬取的时候，可以随机选择 IP 池中的一个 IP 进行。

此时，我们可以为 Scrapy 爬虫项目建立一个下载中间件，在下载中间件中设置好 IP 选择规则，在 settings.py 设置文件中配置好下载中间件，并配置好 IP 池。

首先，在项目核心目录下创建下载中间件文件，文件名可以自定义，只需要配置好即可，如下所示：

```
D:\Python35\myweb\part12\myfirstpjt>cd myfirstpjt\
D:\Python35\myweb\part12\myfirstpjt\myfirstpjt>echo #>middlewares.py
#
```

其实我们已经创建了一个名为 middlewares.py 的 Python 文件，可以将该文件作为下载中间件文件使用。

由于我们需要大量的 IP，此时可以通过代理服务器解决。巧妇难为无米之炊，我们如何找到这些代理服务器呢？读者可以在网上自行搜索，也可以直接使用我们提供的网址进行寻找，在之前的章节中已经详细讲过，即通过 <http://yum.iqianyue.com/proxy> 找到代理 IP。

找到这些代理服务器的 IP 信息后，可以在 settings.py 文件中将这些代理服务器的 IP 设置为 IP 池，如下所示，直接编辑爬虫项目中的 settings.py 文件并添加如下信息：

```
#IP 池设置
IPPOOL=[
    {"ipaddr": "121.33.226.167:3128"},
    {"ipaddr": "118.187.10.11:80"},
    {"ipaddr": "123.56.245.138:808"},
    {"ipaddr": "139.196.108.68:80"},
    {"ipaddr": "36.250.87.88:47800"},
    {"ipaddr": "123.57.190.51:7777"},
    {"ipaddr": "171.39.26.176:8123"}
]
```

此时，IPPOOL 就是对应的代理服务器的 IP 池，外层通过列表的形式存储，里层通过字典的形式存储。

设置好 IP 池后，我们需要编写下载中间件文件。

在 Scrapy 中，与代理服务器设置相关的下载中间件是 HttpProxyMiddleware，同样，在 Scrapy 官方文档中，HttpProxyMiddleware 对应类为：

```
class scrapy.contrib.downloadermiddleware.httpproxy.HttpProxyMiddleware
```

所以，在编辑下载中间件的时候需要导入 `scrapy.contrib.downloadermiddleware.httpproxy` 下的 `HttpProxyMiddleware`。

在这里我们会编辑刚刚创建的下载中间件文件 `middlewares.py`，需要在下载中间件文件中写入具体的程序实现，如下所示，关键的地方已经给出注释：

```
#middlewares 下载中间件
# 导入随机数模块，目的是随机挑选一个 IP 池中的 IP
import random
# 从 settings 文件 (myfirstpjt.settings 为 settings 文件的地址) 中导入设置好的 IPPOOL
from myfirstpjt.settings import IPPOOL
# 导入官方文档中 HttpProxyMiddleware 对应的模块
from scrapy.contrib.downloadermiddleware.httpproxy import HttpProxyMiddleware

class IPPOOLS(HttpProxyMiddleware):
    # 初始化方法
    def __init__(self, ip=''):
        self.ip=ip
    # process_request() 方法，主要进行请求处理
    def process_request(self, request, spider):
        # 先随机选择一个 IP
        thisip=random.choice(IPPOOL)
        # 输出当前选择的 IP，便于调试观察
        print("当前使用的 IP 是: "+thisip["ipaddr"])
        # 将对应的 IP 实际添加为具体的代理，用该 IP 进行爬取
        request.meta["proxy"]="http://"+thisip["ipaddr"]
```

编写好下载中间件之后，此时在 Scrapy 项目中，`middlewares.py` 文件只是一个普通的 Python 程序，并不会默认为下载中间件。如果想让 `middlewares.py` 文件作为项目的下载中间件文件，还需要在 `settings.py` 文件中进行相应配置。

在 `settings.py` 文件中，与下载中间件相关的配置信息默认如下：

```
#DOWNLOADER_MIDDLEWARES = {
#    'myfirstpjt.middlewares.MyCustomDownloaderMiddleware': 543,
#}
```

所以现在需要对这一部分配置信息进行如下修改：

```
DOWNLOADER_MIDDLEWARES = {
    'myfirstpjt.middlewares.MyCustomDownloaderMiddleware': 543,
    'scrapy.contrib.downloadermiddleware.httpproxy.HttpProxyMiddleware':123,
    'myfirstpjt.middlewares.IPPOOLS':125
}
```

在上面的代码中，我们首先根据官方文档配置了 `'scrapy.contrib.downloadermiddleware.httpproxy.HttpProxyMiddleware'` 这一项内容，随后在配置了下载中间件对应所在的地方以及其内部的类，格式为：“下载中间件所在目录.下载中间件文件名.下载中间件内部要使用的类”，所以配置为 `'myfirstpjt.middlewares.IPPOOLS'`，“myfirstpjt”为下载中间件所在目录，

“middlewares”为下载中间件文件名，“IPPOOLS”为下载中间件内部要使用的类。

配置好之后，刚才所编写的 Python 文件就正式成了 Scrapy 项目中的下载中间件文件，在运行该项目中的爬虫的时候，就可以使用下载中间件进行网页的下载。

在 CMD 命令行中运行爬虫 weisuen，具体结果如下：

```
D:\Python35\myweb\part12\myfirstpjt\myfirstpjt>scrapy crawl weisuen --nolog
当前使用的 IP 是: 171.39.26.176:8123
当前使用的 IP 是: 123.56.245.138:808
当前使用的 IP 是: 123.57.190.51:7777
当前使用的 IP 是: 171.39.26.176:8123
当前使用的 IP 是: 123.57.190.51:7777
当前使用的 IP 是: 123.56.245.138:808
以下将显示爬取的网址的标题
[<Selector xpath='/html/head/title/text()' data='告别“政策市”，才能铲除房产谣言土壤
| 房产 | 辟谣 | 政策_新浪新闻 '>]
当前使用的 IP 是: 36.250.87.88:47800
当前使用的 IP 是: 121.33.226.167:3128
以下将显示爬取的网址的标题
[<Selector xpath='/html/head/title/text()' data='印度怕了吗？疑歼 11D 歼 16 与歼 20 同
在高原测试_高清图集_新浪网 '>]
当前使用的 IP 是: 118.187.10.11:80
以下将显示爬取的网址的标题
[<Selector xpath='/html/head/title/text()' data='任性的丽江周末已经开始下雪_高清图
集_新浪网 '>]
```

可以看到，此时会随机选择 IP 池中的 IP 对网页进行爬取。这样，对方的网站服务器即使屏蔽了我们的其中一个 IP，我们还能够直接使用其他的 IP 进行自动爬取。

#### 4. 使用用户代理池

我们知道，网站服务器可以识别爬行时候的用户代理（User-Agent）信息，通过用户代理信息可以判断出我们使用的是什么浏览器、什么形式的客户端等。

所以，对方的网站服务器可以根据我们的 User-Agent 信息，对我们的爬行行为进行分析，以此来实现反爬虫处理。

作为爬虫方，显然不希望就这样被封禁，为了避免这一类的禁止，我们可以使用用户代理池进行处理。

这一种处理方法与 IP 代理池的处理方法类似，我们可以搜集多种浏览器的信息，以此建立一个用户代理池，然后再建立一个下载中间件，在下载中间件中设置每次随机选择用户代理池中的一个用户代理进行爬行。

与 IP 代理池的处理方法不同的地方是，此时我们要使用的下载中间件类型为 UserAgentMiddleware，而在 IP 代理池中使用的下载中间件类型为 HttpProxyMiddleware。

首先，需要在 settings.py 配置文件中设置好用户代理池，代理池的名称可以自定义，如下所示：

```
# 用户代理 (user-agent) 池设置
UAPOOL=[
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/49.0.2623.22 Safari/537.36 SE 2.X MetaSr 1.0",
    "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0",
    "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/536.5"
```

通过以上代码我们设置了一个名为 UAPOOL 的用户代理池，接下来我们创建一个下载中间件文件，如下所示：

```
D:\Python35\myweb\part12\myfirstpjt\myfirstpjt>echo #>uamid.py
#
```

创建好之后，可以编辑该下载中间件文件中的代码来实现对应功能。

在 Scrapy 中，IP 代理池中使用的下载中间件类型为 `HttpProxyMiddleware`，对应的类为：

```
class scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware
```

所以，我们在下载中间件文件中需要从 `scrapy.contrib.downloadermiddleware.useragent` 导入 `UserAgentMiddleware`，然后再进行相应代码的编写。

可以在下载中间件文件中写上如下代码，关键的地方已经给出注释：

```
#uamid 下载中间件
import random
from myfirstpjt.settings import UAPOOL
from scrapy.contrib.downloadermiddleware.useragent import UserAgentMiddleware

class Uamid(UserAgentMiddleware):
    def __init__(self,ua=''):
        self.ua=ua
    def process_request(self,request,spider):
        thisua=random.choice(UAPOOL)
        print("当前使用的 user-agent 是: "+thisua)
        request.headers.setdefault('User-Agent',thisua)
```

随后，还需要在 `settings.py` 文件中将该 Python 文件设置为 Scrapy 中的下载中间件，我们可以打开 `settings.py` 文件，找到有关下载中间件设置的地方（`DOWNLOADER_MIDDLEWARES`），修改为如下代码：

```
# Enable or disable downloader middlewares
# See http://scrapy.readthedocs.org/en/latest/topics/downloader-middleware.html
DOWNLOADER_MIDDLEWARES = {
    #'myfirstpjt.middlewares.MyCustomDownloaderMiddleware': 543,
    'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware':2,
    'myfirstpjt.uamid.Uamid':1
}
```

设置好之后，我们可以运行该 Scrapy 项目下的爬虫文件了，此时我们可以运行该项目下



的爬虫文件 weisuen，如下所示：

```
D:\Python35\myweb\part12\myfirstpjt\myfirstpjt>scrapy crawl weisuen --nolog
当前使用的 user-agent 是: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/536.5
当前使用的 user-agent 是: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:48.0) Gecko/20100101
Firefox/48.0
当前使用的 user-agent 是: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:48.0) Gecko/20100101
Firefox/48.0
当前使用的 user-agent 是: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/536.5
当前使用的 user-agent 是: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:48.0) Gecko/20100101
Firefox/48.0
当前使用的 user-agent 是: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:48.0) Gecko/20100101
Firefox/48.0
以下将显示爬取的网址的标题
[<Selector xpath='/html/head/title/text()' data='任性的丽江周末已经开始下雪_高清图集_新浪网 '>]
以下将显示爬取的网址的标题
[<Selector xpath='/html/head/title/text()' data='告别“政策市”，才能铲除房产谣言土壤|房产|辟谣|政策_新浪新闻 '>]
以下将显示爬取的网址的标题
[<Selector xpath='/html/head/title/text()' data='印度怕了吗？疑歼11D歼16与歼20同在高原测试_高清图集_新浪网 '>]
```

可以看到，此时会通过下载中间件自动的随机切换用户代理进行网页爬取，这样，对方的网站服务器相对来说就比较难通过识别用户代理来屏蔽我们的爬虫。

## 5. 其他方法

除了上述方法之外，我们还可以使用一些其他方法避免被 ban，比如使用谷歌 Cache、使用分布式爬行等方式。使用分布式爬行的方式中，比较常用的可以通过 scrapinghub 旗下的框架进行，之前在第 10 章中已经介绍过，可以在 <https://scrapinghub.com/crawlera/> 中建立自己的 Scrapy 爬虫项目，如图 12-17 所示为 scrapinghub 旗下 crawlera 项目的首页。

以上我们介绍了避免被 ban 的常见几种方法，当然，我们使用这些方法对别人的网站进行爬取的时候，需要遵守道德，遵守 robots 协议，不利用这些技巧进行违法的事项。在遵守道德和法律的前提下，希望读者利用这些技巧可以畅快地对需要的网络数据进行爬取。



图 12-17 crawlera 项目首页

## 12.12 小结

1) 我们可以通过 scrapy startproject -h 调出 startproject 的帮助信息，在这里可以看到 scrapy

startproject 具体可以添加哪些参数。

2) 如果我们想要删除某个爬虫项目, 我们可以直接删除该爬虫项目对应的文件夹即可实现。

3) 可以使用 genspider 命令来创建 Scrapy 爬虫文件, 这是一种快速创建爬虫文件的方式。

4) 爬虫的测试比较麻烦, 所以在 Scrapy 中使用合同 (contract) 的方式对爬虫进行测试。

5) 通过 parse 命令, 我们可以实现获取指定的 URL 网址, 并使用对应的爬虫文件进行处理和分析。

6) 使用 Scrapy 中的 Item 对象可以保存爬取到的数据, 相当于存储爬取到的数据的容器。

7) 在 Scrapy 中, 经常会使用 XPath 表达式进行数据的筛选和提取。

8) 我们经常使用 XMLFeedSpider 去处理 RSS 订阅信息。RSS 是一种信息聚合技术, 可以让信息的发布和共享更为高效、便捷。同样, RSS 是基于 XML 标准的。

9) 在 Scrapy 项目中, 我们主要可以通过以下方法来避免被禁止: 禁止 Cookie、设置下载延时、使用 IP 池、使用用户代理池或其他方法。

## 13.2 常用的 Scrapy 组件

通过上一节的学习, 我们已经知道 Scrapy 中基本有哪些组件。从总体上看, Scrapy 的架构, 那么这些组件各自具体有什么作用呢?

下面我们将分别对这些组件进行详细讲解。

### 1. Scrapy 引擎

Scrapy 引擎是整个 Scrapy 架构的核心, 负责控制整个数据处理的流程。



## Chapter 13

## 第 13 章

## Scrapy 核心架构

在上一章中，我们已经学习了如何使用 Scrapy 框架来编写爬虫项目，那么具体 Scrapy 框架中底层是如何架构的呢？Scrapy 主要拥有哪些组件，爬虫具体的实现过程又是怎样的呢？

为了更深入的了解 Scrapy 的相关知识，我们需要对 Scrapy 的架构以及 Scrapy 中常见的组件进行了解，并熟悉 Scrapy 爬虫项目的工作流程。

本章中，我们将对这些 Scrapy 的理论知识进行详细分析。

### 13.1 初识 Scrapy 架构

Scrapy 的架构在 Scrapy 官方文档中有详尽的描述，为了方便读者理解，笔者重画了 Scrapy 架构图（架构图中不包含 workflow 部分），并结合该架构图绘制了对应的 Scrapy 工作流图。

如图 13-1 所示，为 Scrapy 的架构图示，其中处于中心位置的 Scrapy 引擎为 Scrapy 框架架构的核心。

从图中我们可以看得到，Scrapy 中的组件主要包括：

- 1) Scrapy 引擎
- 2) 调度器
- 3) 下载器
- 4) 下载中间件
- 5) 蜘蛛（也叫作爬虫，一个 Scrapy 项目下可能会有多个 Spiders 爬虫文件）
- 6) 爬虫中间件
- 7) 实体管道

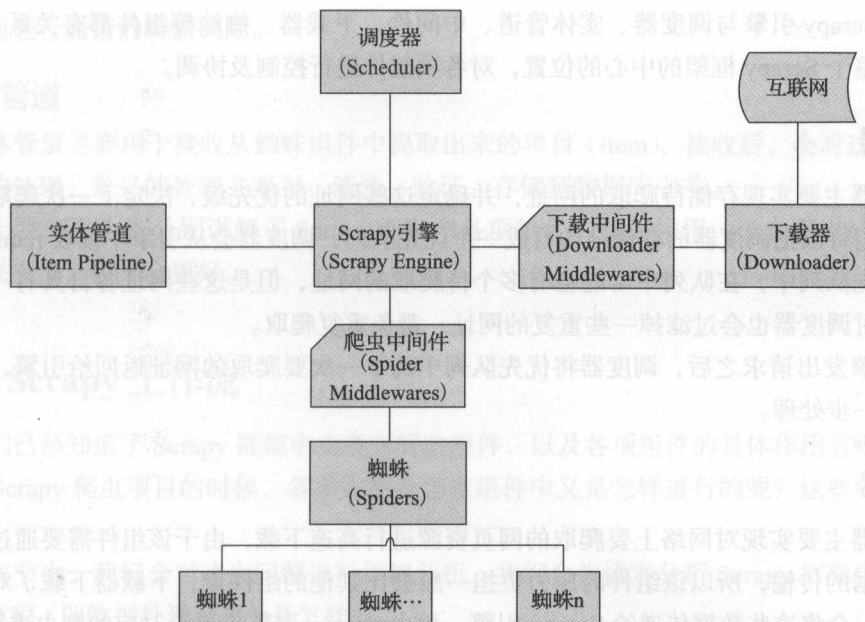


图 13-1 Scrapy 的架构图示

其中，Scrapy 引擎为整个架构的核心，调度器、实体管道、下载器和蜘蛛等组件都通过 Scrapy 引擎来调控。在 Scrapy 引擎和下载器之间，可以通过一个叫下载中间件的组件进行信息的传递，在下载中间件中，可以插入一些自定义的代码来轻松扩展 Scrapy 的功能，在上一章中我们就通过下载中间件实现 IP 池和用户代理池的应用。同样，Scrapy 引擎和蜘蛛之间，也可以通过一个叫爬虫中间件的组件进行爬虫与 Scrapy 引擎间的信息传递，我们也可以在爬虫中间件中添加一些自定义代码来轻松扩展 Scrapy 的功能。在一个 Scrapy 项目中，可以存在多个爬虫文件，在运行的时候，我们可以指定运行某一个爬虫文件，在图中我们也可以看到有多个蜘蛛。通过 Scrapy 引擎可以调度下载器进行下载互联网中的网页数据，以传递给爬虫（蜘蛛）文件进行对应的处理，具体的工作流程我们可以在 13.3 节中详细了解。

## 13.2 常用的 Scrapy 组件详解

通过上一节的学习，我们已经知道 Scrapy 中基本有哪些组件，从总体上了解了 Scrapy 的架构，那么这些组件各自具体有什么作用呢？

下面我们将分别对这些组件进行详细讲解。

### 1. Scrapy 引擎

Scrapy 引擎是整个 Scrapy 架构的核心，负责控制整个数据处理流程，以及触发一些事



务处理。Scrapy 引擎与调度器、实体管道、中间件、下载器、蜘蛛等组件都有关系，Scrapy 引擎处于整个 Scrapy 框架的中心的位置，对各项组件进行控制及协调。

## 2. 调度器

调度器主要实现存储待爬取的网址，并确定这些网址的优先级，决定下一次爬取哪个网址等。我们可以把调度器的存储结构看成一个优先队列，调度器会从引擎中接收 request 请求并存入优先队列中，在队列中可能会有多个待爬取的网址，但是这些网址各自具有一定的优先级，同时调度器也会过滤掉一些重复的网址，避免重复爬取。

当引擎发出请求之后，调度器将优先队列中的下一次要爬取的网址返回给引擎，以供引擎进行进一步处理。

## 3. 下载器

下载器主要实现对网络上要爬取的网页资源进行高速下载，由于该组件需要通过网络进行大量数据的传输，所以该组件的压力负担一般会比其他的组件重。下载器下载了对应的网页资源后，会将这些数据传递给 Scrapy 引擎，再由 Scrapy 引擎传递给对应的爬虫进行处理。

## 4. 下载中间件

下载中间件是处于下载器和 Scrapy 引擎之间的一个特定的组件，主要用于对下载器和 Scrapy 引擎之间的通信进行处理，在下载中间件中，可以加入自定义代码，轻松地实现 Scrapy 功能的扩展，我们在下载中间件中加入的自定义代码，会在 Scrapy 引擎与下载器通信的时候调用。

上一章中我们已经具体的使用过下载中间件来实现 IP 池和用户代理池的相关功能。

## 5. 蜘蛛

蜘蛛 (Spider) 组件，也叫作爬虫组件，该组件是 Scrapy 框架中爬虫实现的核心。在一个 Scrapy 项目中，可以有多个蜘蛛，每个蜘蛛可以负责一个或多个特定的网站。蜘蛛组件主要负责接收 Scrapy 引擎中的 response 响应（这些响应具体是下载器从互联网中得到的响应然后传递到 Scrapy 引擎中的），在接收了 response 响应之后，蜘蛛会对这些 response 响应进行分析处理，然后可以提取出对应的关注的数据，也可以提取出接下来需要处理的新网址等信息。

## 6. 爬虫中间件

爬虫中间件是处于 Scrapy 引擎与爬虫组件之间的一个特定的组件，主要用于对爬虫组件和 Scrapy 引擎之间的通信进行处理。同样，在爬虫中间件中可以加入一些自定义代码，很轻松地实现 Scrapy 功能的扩展。在爬虫中间件中加入的自定义代码，会在 Scrapy 引擎与爬虫

组件之间进行通信的时候调用。

## 7. 实体管道

实体管道主要用于接收从蜘蛛组件中提取出来的项目 (item)，接收后，会对这些 item 进行对应的处理，常见的处理主要有：清洗、验证、存储到数据库中等。

以上我们为大家分别讲解了 Scrapy 框架中各项组件的具体作用，读者可以结合 13.1 节中的图来阅读，效果会更好。

## 13.3 Scrapy 工作流

我们已经知道了 Scrapy 框架中主要有哪些组件，以及各项组件的具体作用有哪些，那么在运行 Scrapy 爬虫项目的时候，各项数据处理在组件中又是怎样进行的呢？这些组件会如何配合呢？

在本节中，我们会对这个问题进行详细分析，我们会为读者分析 Scrapy 框架中各项组件的工作流程（即数据处理流程）是怎样的。

在此，笔者依据 13.1 节中的组件图绘制了 Scrapy 框架中数据处理流程的具体图示，如图 13-2 所示。

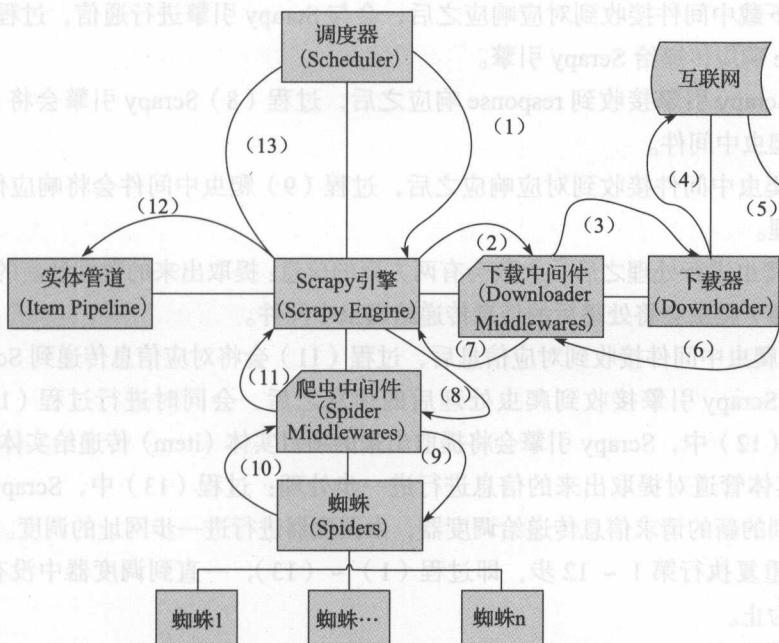


图 13-2 Scrapy 框架中数据处理流程的具体图示

图中 (1) ~ (13) 为一个较完整 Scrapy 中的数据流向，总共为 13 个过程，箭头所指的

方向即为数据流向的方向，接下来我们分别为大家分析（1）~（13）过程中的每一个过程的具体含义。

首先，Scrapy 引擎会将爬虫文件中设置的要爬取的起始网址（默认在 `start_urls` 属性中设置）传递到调度器中。随后，依次进行图中（1）~（13）过程。

第 1 步：过程（1）中，主要将下一次要爬取的网址传递给 Scrapy 引擎，调度器是一个优先队列，里面可能存储着多个要爬取的网址（当然也可能只有一个网址），调度器会根据各网址的优先级分析出下一次要爬取的网址，然后再传递给 Scrapy 引擎。

第 2 步：Scrapy 引擎接收到（1）中传过来的网址之后，过程（2）Scrapy 引擎主要将网址传递给下载中间件。

第 3 步：下载中间件接收到 Scrapy 引擎传过来的网址之后，过程（3）中下载中间件会将对应的网址传递给下载器。

第 4 步：然后，下载器接收到对应要下载的网址，然后过程（4）会向互联网中对应的网址发送 `request` 请求，进行网页的下载。

第 5 步：互联网中对应的网址接收到 `request` 请求之后，会有相应的 `response` 响应，随后在过程（5）中将响应返回给下载器。

第 6 步：下载器接收到响应之后，即完成了对应网页的下载，随后过程（6）会将对应的响应传送给下载中间件。

第 7 步：下载中间件接收到对应响应之后，会与 Scrapy 引擎进行通信，过程（7）会将对应的 `response` 响应传递给 Scrapy 引擎。

第 8 步：Scrapy 引擎接收到 `response` 响应之后，过程（8）Scrapy 引擎会将 `response` 响应信息传递给爬虫中间件。

第 9 步：爬虫中间件接收到对应响应之后，过程（9）爬虫中间件会将响应传递给对应的爬虫进行处理。

第 10 步：爬虫进行处理之后，大致会有两方面的信息：提取出来的数据和新的请求信息。然后，过程（10）爬虫会将处理后的信息传递给爬虫中间件。

第 11 步：爬虫中间件接收到对应信息后，过程（11）会将对应信息传递到 Scrapy 引擎。

第 12 步：Scrapy 引擎接收到爬虫处理后的信息之后，会同时进行过程（12）和过程（13）。在过程（12）中，Scrapy 引擎会将提取出来的项目实体（`item`）传递给实体管道（`Item Pipeline`），由实体管道对提取出来的信息进行进一步处理；过程（13）中，Scrapy 引擎会将爬虫处理后得到的新的请求信息传递给调度器，由调度器进行进一步网址的调度。

随后，再重复执行第 1~12 步，即过程（1）~（13），一直到调度器中没有网址调度或者异常退出为止。

以上我们分析了 Scrapy 框架中各项组件的工作流程，此时，我们对 Scrapy 框架中数据处理的过程就有了比较详细的了解，理清了该过程之后，我们在编写 Scrapy 爬虫项目的时候，思路就能更加清晰。

## 13.4 小结

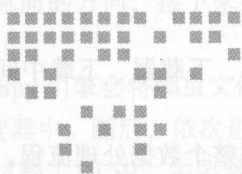
1) Scrapy 中的组件主要包括: Scrapy 引擎、调度器、下载器、下载中间件、爬虫、爬虫中间件、实体管道。

2) Scrapy 引擎是整个 Scrapy 架构的核心, 负责控制整个数据处理流程, 以及负责触发一些事务处理。Scrapy 引擎与调度器、实体管道、中间件、下载器、蜘蛛等组件都有关系, Scrapy 引擎处于整个 Scrapy 框架的中心的位罝, 对各项组件进行控制及协调。

3) 在一个 Scrapy 项目中, 可以有多个蜘蛛, 每个蜘蛛可以负责一个或一些特定的网站。蜘蛛组件主要负责接收 Scrapy 引擎中的 response 响应(这些响应具体是下载器从互联网中得到的响应然后传递到 Scrapy 引擎中的), 接收了 response 响应之后, 蜘蛛会对这些 response 响应进行分析处理, 然后可以提取出对应的关注的数椐, 也可以提取出接下来需要处理的新网址等信息。

4) 爬虫中间件是处于 Scrapy 引擎与爬虫组件之间的一个特定的组件, 主要用于对爬虫组件和 Scrapy 引擎之间的通信进行处理。同样, 在爬虫中间件中可以加入一些自定义代码, 很轻松地实现 Scrapy 功能的扩展, 在爬虫中间件中加入的自定义代码, 会在 Scrapy 引擎与爬虫组件之间进行通信的时候调用。





## Scrapy 中文输出与存储

有的时候，我们使用 Scrapy 爬虫爬取的信息中可能会含有中文信息，而在 Python 中，处理中文信息的时候需要一定的技巧，否则可能会出现无法显示中文或乱码的情况，所以在本章中，我们会为大家讲解 Scrapy 爬虫中中文信息的处理，包括如何在命令行中直接输出爬取的中文信息以及如何存储爬取到的中文信息到普通文件以及 JSON 文件中。

### 14.1 Scrapy 的中文输出

其实，在 Python3.X 中，中文信息直接输出处理方面已经做得相对比较好了，而在 Python2.X 中，直接输出中文信息常常会遇到一些问题，所以需要一些处理技巧。考虑到有些朋友使用的还是 Python2.X，所以在本节中，我们会为大家讲解一下在 Python2.X 中如何直接输出中文信息。

比如，我们可以先在 Python2.X 的环境中编写相应的 Scrapy 爬虫程序。

首先，创建一个名为 mypyjt 的爬虫项目，再在项目下创建一个基于 basic 模板的名为 myspyd 的爬虫文件。

然后，进入 mypyjt 爬虫项目，编写修改 items.py 文件，修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-
# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html
```

```
import scrapy
class MypjtItem(scrapy.Item):
    # define the fields for your item here like:
    # 定义title, 用来存储网页标题信息
    title=scrapy.Field()
```

编写好 items.py 文件之后, 我们再来编写 mypjt 项目中的 spiders/myspd.py 爬虫文件, 修改为如下所示, 同样, 关键部分已给出注释:

```
# -*- coding: utf-8 -*-
import scrapy
# 导入 items 文件中的 MypjtItem
from mypjt.items import MypjtItem
class MyspdSpider(scrapy.Spider):
    name = "myspd"
    allowed_domains = ["sina.com.cn"]
    start_urls = (
        # 定义要爬取的起始网址为新浪首页
        'http://www.sina.com.cn/',
    )
    def parse(self, response):
        # 初始化 item
        item=MypjtItem()
        # 通过 XPath 表达式提取该网页中的标题信息
        item["title"]=response.xpath("/html/head/title").extract()
        # 输出提取到的标题信息
        print item["title"]
```

编写好爬虫文件之后, 接下来我们可以在命令行中运行爬虫文件。在命令行中进入爬虫项目 mypjt 所在的目录, 然后通过 scrapy crawl 运行该爬虫项目下的 myspd 爬虫文件, 为了避免输出的信息过多, 可以通过 --nolog 参数控制不显示日志信息, 运行结果如下所示:

```
D:\Python27\mypjt>scrapy crawl myspd --nolog
[u'<title>\u65b0\u6d6a\u9996\u9875</title>']
```

可以看到, 此时输出为 u'<title>\u65b0\u6d6a\u9996\u9875</title>', 而新浪首页的标题是“新浪首页”, 为什么这里信息会不对应呢? 是因为此时输出的信息中含有中文, 出现了相应的编码问题, 如果此时想直接输出中文信息, 我们需要对要输出的这个信息进行 encode 编码。

所以, 可以对项目中的 spiders/myspd.py 进行相应修改:

```
# -*- coding: utf-8 -*-
import scrapy
from mypjt.items import MypjtItem
class MyspdSpider(scrapy.Spider):
    name = "myspd"
    allowed_domains = ["sina.com.cn"]
```

```

start_urls = (
    'http://www.sina.com.cn/',
)
def parse(self, response):
    item=MypjtItem()
    item["title"]=response.xpath("/html/head/title").extract()
    #print item["title"]
    # item["title"] 是一个列表，所以我们可以通过 for 循环遍历出该列表中的元素
    for i in item["title"]:
        # 对遍历出来的标题信息进行 encode("gbk") 编码
        print i.encode("gbk")

```

修改好对应的爬虫文件之后，我们可以再次运行该项目中的爬虫，运行结果如下所示：

```

D:\Python27\mypjt>scrapy crawl myspd --nolog
<title> 新浪首页 </title>

```

此时可以发现，已经成功输出了对应的爬取到的新浪首页的标题，标题为中文信息，我们成功通过 `encode` 处理了编码的问题。

以上，就是在 Python2.X 环境下的 Scrapy 爬虫中直接输出中文信息的处理方式，如果 `encode("gbk")` 出现乱码，可以将 `encode("gbk")` 换成 `encode("utf-8")` 尝试解决。

而在 Python3.X 中，直接输出中文信息方面的能力已经比较完善。在本书中，我们会以新版的 Python3.X 作为主线进行讲解，上面讲解 Python2.X 环境下 Scrapy 爬虫中直接输出中文信息的处理办法目的是为了兼顾使用 Python2.X 的读者。

接下来，我们可以稍微看一下在 Python3.X 环境下运用 Scrapy 框架直接输出中文信息的能力。

以下的程序将在 Python3.X 环境下运行。

同样，首先创建一个爬虫项目 `mypjt`，如下所示：

```

D:\Python35\myweb\part13>scrapy startproject mypjt
New Scrapy project 'mypjt', using template directory 'd:\python35\lib\site-packages\scrapy\templates\project', created in:
    D:\Python35\myweb\part13\mypjt

You can start your first spider with:
    cd mypjt
    scrapy genspider example example.com

```

在创建好了对应的项目之后，可以在该项目下创建一个基于 basic 爬虫模板的爬虫文件 `weisuen.py`，如下所示：

```

D:\Python35\myweb\part13\mypjt>scrapy genspider -t basic weisuen sina.com.cn --nolog
Created spider 'weisuen' using template 'basic' in module:
    mypjt.spiders.weisuen

```

随后，按照正常的思路去编写 `items.py` 文件，如下所示：

```
# -*- coding: utf-8 -*-
import scrapy
class MypjtItem(scrapy.Item):
    # define the fields for your item here like:
    name = scrapy.Field()
    title=scrapy.Field()
```

接下来，还需要修改与编写爬虫文件 spiders/weisuen.py，如下所示：

```
# -*- coding: utf-8 -*-
import scrapy
from mypjt.items import MypjtItem
class WeisuenSpider(scrapy.Spider):
    name = "weisuen"
    allowed_domains = ["sina.com.cn"]
    start_urls = (
        # 设置起始网址为新浪新闻下的某个新闻网页
        'http://tech.sina.com.cn/d/s/2016-09-17/doc-ixvyqwa3324638.shtml',
    )

    def parse(self, response):
        item=MypjtItem()
        # 通过 XPath 表达式提取网页中的标题信息
        item["title"]=response.xpath("/html/head/title/text()")
        # 直接输出，在 Python3.X 中，虽然包含中文信息，但直接输出即可
        print(item["title"])
```

编写好对应的爬虫文件之后，接下来我们可以在命令行中进入该爬虫所在的目录，然后运行该爬虫项目下的 weisuen 爬虫，进行网页的爬取，运行结果如下所示：

```
D:\Python35\myweb\part13\mypjt>scrapy crawl weisuen --nolog
[<Selector xpath='/html/head/title/text()' data=' "神舟十一号" 载人飞船将于 10 月中旬发射 | 载人飞船 | 神舟十一号 | 航天员_新浪科技 '>]
```

此时我们可以看到，在命令行中直接输出了对应的含有中文的标题信息，在 Python3.X 中，编码处理方面的能力相对来说是比较好的。

以上我们为大家讲解了 Scrapy 爬虫中中文输出处理方面的知识。在 Python2.X 环境中，需要通过 encode() 进行相应的编码；在 Python3.X 环境中，可以直接输出。

## 14.2 Scrapy 的中文存储

假如，我们需要将 Scrapy 爬虫中提取到的含有中文的信息存储到某个文件中，那么应该如何实现呢？

在上一章中，我们为大家提到了 Scrapy 架构中基本的组件，我们知道，在 Scrapy 中如果要对提取到的数据进行进一步处理，可以通过 pipelines.py 文件实现。



首先，进入上一节中创建的 Scrapy 爬虫项目 mypjt 对应的文件夹，然后打开 settings.py 文件来配置 pipelines。我们需要在 settings.py 文件中告诉系统 pipelines 文件在哪以及 pipelines 文件里面对应的类是什么，找到 settings.py 文件中关于 pipelines 的设置的部分，默认的设置如下所示。

```
# Configure item pipelines
# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    'mypjt.pipelines.SomePipeline': 300,
}
```

在上面的代码中，'mypjt.pipelines.SomePipeline' 中的 mypjt 为项目名（即 Scrapy 项目的核心目录名），pipelines 代表 mypjt 目录下的 pipelines.py 文件的文件名，SomePipeline 代表对应的 pipelines 文件里的类。所以 'mypjt.pipelines.SomePipeline' 相当于告诉了系统 pipelines 文件在哪，同时又说明了 pipelines 文件里面对应的类是什么，该项设置的格式为“核心目录名.pipelines 文件名.对应类名”。

所以，根据本项目的实际情况，我们将上面的默认配置修改为如下：

```
# Configure item pipelines
# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    # 本项目中在 pipelines 文件里面的类是 MypjtPipeline，接下来会具体看到
    'mypjt.pipelines.MypjtPipeline': 300,
}
```

在 settings.py 中配置好 pipelines 之后，我们需要具体编写 pipelines 文件，打开“D:\Python35\myweb\part13\mypjt\mypjt\pipelines.py”文件，将代码修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-
# 导入 codecs 模块，使用 codecs 直接进行解码
import codecs
# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html
# 定义了 pipelines 里面的类，类名需要与刚才 settings.py 里面设置的类名对应起来
class MypjtPipeline(object):
    # __init__() 为类的初始化方法，开始的时候调用
    def __init__(self):
        # 首先以写入的方式创建或打开一个普通文件用于存储爬取到的数据
        self.file = codecs.open("D:/python35/myweb/part13/mydata1.txt", "wb", encoding="utf-8")
    # process_item() 为 pipelines 中的主要处理方法，默认会自动调用
    def process_item(self, item, spider):
        # 设置每行要写的内容
```

```

l = str(item) + '\n'
# 此处通过 print() 输出, 方便程序的调试
print(l)
# 将对应信息写入文件中
self.file.write(l)
return item
#close_spider() 方法一般在关闭蜘蛛时调用
def close_spider(self, spider):
    # 关闭文件, 有始有终
    self.file.close()

```

编写好对应的 pipelines 文件之后, 在运行爬虫的时候, 会将提取到的数据通过 pipelines 文件进行后续的处理。在这里我们主要将提取到的数据写入指定的文件 mydata1.txt 中, 我们可以运行该项目中的爬虫 weisuen, 结果如下:

```

D:\Python35\myweb\part13\mypjt>scrapy crawl weisuen --nolog
{'title': ['“神舟十一号”载人飞船将于10月中旬发射|载人飞船|神舟十一号|航天员_新浪科技_新浪网']}

```

可以看到, 此时 pipelines 文件中的 print() 直接输出了对应的 title 信息, 方便我们进行调试。打开文件 "D:\python35\myweb\part13\mydata1.txt", 可以看到如图 14-1 所示信息。

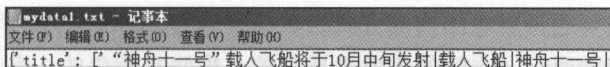


图 14-1 将数据存储到普通文件中图示

此时我们发现, 提取到的信息已经成功写入到了普通文本文件 “mydata1.txt” 中。

在这里需要注意的是, 在 pipelines 文件中设置的要写入的每一行的内容是 `l = str(item) + '\n'`, 其中需要把 item 的类型通过 str() 转换为字符串类型, 这样类型才能对应起来。如果在 pipelines 中, 我们把 `l = str(item) + '\n'` 换成:

```
l = item + '\n'
```

则会出现下面的错误:

```
TypeError: unsupported operand type(s) for +: 'MypjtItem' and 'str'
```

所以, 这里需要注意保证类型的一致性。

### 14.3 输出中文到 JSON 文件

JSON 是一种在编程中常用的数据格式, 属于一种轻量级的数据交换格式。

在本节中, 我们会讲解如何将 Scrapy 爬虫爬取到的含有中文的数据存储为 JSON 格式的

文件。

JSON 格式的数据的书写方式一般为：“名称：值对”，各项数据之间可以通过逗号隔开。JSON 数据常见的基本存储结构有数组和对象两种。

比如，我们可以将“苹果、梨子、葡萄”三个数据存储为数组结构的 JSON 文件，存储方式如下：

```
[“苹果”，“梨子”，“葡萄”]
```

这种数组结构的存储方式，对应的值是通过索引的方式进行获取的，对应关系如下：

```
0-- “苹果”
1-- “梨子”
2-- “葡萄”
```

如果要获取数据“葡萄”，可以通过下标索引 [2] 进行获取。

除了将数据存储为数组结构的 JSON 文件之外，我们还可以将数据存储为对象结构的 JSON 文件。

对象结构的 JSON 文件中的数据一般通过 {} 括起来，里面的数据一般是键值对的形式，键与值之间通过“:”分开，各项数据之间通过“,”隔开，基本的格式是：“{key1:value1, key2:value2, key3:value3}”。

所以，假如我们要通过对象结构的方式将数据存储为 JSON 文件，可以按照上面的基本格式进行。比如我们需要将“小明的头发是黑色的，皮肤是黄色的，身高 170”等信息存储为对象结构的 JSON 文件，可以存储为：

```
{“姓名”：“小明”，“头发”：“黑色”，“皮肤”：“黄色”，“身高”：170}
```

如果要获取小明的身高信息，我们可以通过身高的键进行获取：

```
对应对象. “身高”
```

获取到值 170，即获取到小明身高的具体的值，如果要获取其他项，同样按照这种方式获取即可。

JSON 格式的数据使用起来非常方便，所以我们可以将 Scrapy 爬虫爬取到的数据存储为 JSON 格式的文件，但是在处理中文的时候需要一定的技巧，所以在此我们会为大家讲解如何将爬取到的含有中文信息的数据存储为 JSON 格式文件。

在这里我们主要进行的是爬取数据的后续处理，所以主要编写与修改的是项目中的 pipelines 文件。

打开本章第 1 节中在 Python3.X 环境下创建的 mypyjt 爬虫项目对应文件夹中的 pipelines 文件，然后修改为如下所示，关键部分已给出注释。

```
# -*- coding: utf-8 -*-
import codecs
```

```

# 因为要进行 JSON 文件的处理，所以导入 json 模块
import json
# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html
class MypjtPipeline(object):
    def __init__(self):
        # 以写入的方式创建或打开一个 json 格式（后缀名为 .json）的文件
        self.file = codecs.open("D:/python35/myweb/part13/mydata1.json", "wb",
                                encoding="utf-8")
    def process_item(self, item, spider):
        # 通过 dict(item) 将 item 转化成一个字典
        # 然后通过 json 模块下的 dumps() 处理字典数据
        i=json.dumps(dict(item))
        # 得到的数据后加上 "\n" 换行符形成要写入的一行数据
        line = i + '\n'
        # 在此进行直接输出，方便调试，实际的时候输出这一行可以去掉
        print(line)
        # 写入数据到 json 文件中
        self.file.write(line)
        return item
    def close_spider(self, spider):
        # 关闭文件，有始有终
        self.file.close()

```

然后，我们可以在命令行中运行爬虫项目中的 weisuen 爬虫，结果如下所示：

```

D:\Python35\myweb\part13\mypjt>scrapy crawl weisuen --nolog
{"title": ["\u201c\u795e\u821f\u5341\u4e00\u53f7\u201d\u8f7d\u4eba\u98de\u8239\u5c06\u4e8e10\u6708\u4e2d\u65ec\u53d1\u5c04\u8f7d\u4eba\u98de\u8239\u795e\u821f\u5341\u4e00\u53f7\u822a\u5929\u5458_\u65b0\u6d6a\u79d1\u6280\u65b0\u6d6a\u7f51"]}

```

可以看到，爬取到的数据由于涉及中文信息，所以在此直接输出编码有一些问题，那么再存储到的 json 文件中，是否也会有编码问题呢？我们打开刚刚存储的对应的 json 文件 "D:/python35/myweb/part13/mydata1.json"，内容如图 14-2 所示：

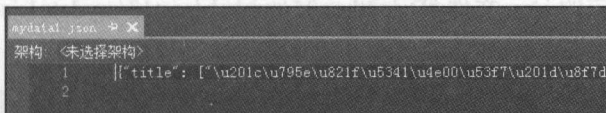


图 14-2 将含有中文信息的数据存储到 json 文件有编码问题时图示

我们发现此时编码仍然是有问题的，那么涉及中文信息的时候，我们应该如何解决编码问题呢？在此我们为大家整理了解决存储中文信息到 json 文件中的技巧，可以通过以下方式解决。

我们需要改进一下 pipeline 文件，如下所示，关键改动部分已给出注释：



```

# -*- coding: utf-8 -*-
import codecs
import json

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

class MypjtPipeline(object):
    def __init__(self):
        self.file = codecs.open("D:/python35/myweb/part13/mydata1.json", "wb",
                                encoding="utf-8")
    def process_item(self, item, spider):
        # 通过 json 模块下的 dumps() 处理的时候
        # 第二个参数将 ensure_ascii 设置为 False
        i=json.dumps(dict(item), ensure_ascii=False)
        line = i + '\n'
        print(line)
        self.file.write(line)
        return item
    def close_spider(self, spider):
        self.file.close()

```

在上面的程序中，因为在进行 `json.dumps()` 序列化的时候，中文信息会默认使用 ASCII 编码，显然此时通过 ASCII 编码无法得到我们想要的显示，所以需要把 `ensure_ascii` 参数设置为 `False`，不使用 ASCII 进行编码，这样就可以处理中文信息了。

我们可以在命令行中再次运行 mypjt 爬虫项目中的 weisuen 爬虫，结果如下所示：

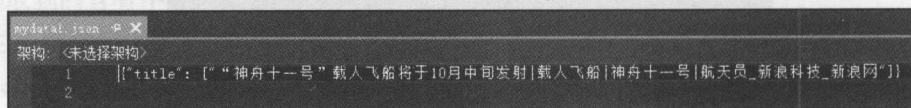
```

D:\Python35\myweb\part13\mypjt>scrapy crawl weisuen --nolog
{"title": [" "神舟十一号" 载人飞船将于 10 月中旬发射 | 载人飞船 | 神舟十一号 | 航天员_新浪科技_新浪网 "]}

```

可以看到，此时直接输出的部分中文信息已经能成功显示了，那么存储到 json 文件中的数据是否也已经正常了呢？

我们可以打开对应的 json 文件 "D:/python35/myweb/part13/mydata1.json"，结果如图 14-3 所示：



```

mydata1.json
架构: 未选择架构
1  [{"title": [" "神舟十一号" 载人飞船将于10月中旬发射 | 载人飞船 | 神舟十一号 | 航天员_新浪科技_新浪网"]}
2

```

图 14-3 将含有中文信息的数据存储到 json 文件中并解决编码问题图示

可以看到，此时存储到的 json 文件中的 JSON 数据也已经能够正常显示中文信息了。此时我们爬取了一个网页的标题信息，那么如果想爬取多条信息以及多个网页，如何实

现呢?

若要爬取多个网页中和多条信息, 可以进行如下修改:

首先在 items 文件中增加一项要爬取的信息, 可以新建一个 `key=scrapy.Field()` 来存储获取到的网页中的关键词信息, 新增的代码如下所示:

```
key=scrapy.Field()
```

然后在爬虫文件 `weisuen.py` 中增加要爬取的起始网站, 对应的 `start_urls` 属性修改为:

```
start_urls = (
    'http://tech.sina.com.cn/d/s/2016-09-17/doc-ifyxygwa3324638.shtml',
    "http://sina.com.cn",
)
```

我们新增了一个要爬取的网址, 即新浪首页 (`http://sina.com.cn`)。

由于此时我们不仅需要提取网页中的标题, 还需要提取网页中的关键词等信息, 所以我们需要修改爬虫文件 `weisuen.py` 中的 `parse()` 函数, 将提取信息的部分新增一行以获取网页中的关键词的代码, 如下所示:

```
item["title"]=response.xpath("/html/head/title/text()").extract()
item["key"]=response.xpath("//meta[@name='keywords']/@content").extract()
```

然后运行 `myproj` 项目下的 `weisuen` 爬虫文件, 如下所示:

```
D:\Python35\myweb\part13\myproj>scrapy crawl weisuen --nolog
{"title": [" "神舟十一号" 载人飞船将于 10 月中旬发射 | 载人飞船 | 神舟十一号 | 航天员 _ 新浪科技 _ 新浪网"], "key": [" 载人飞船, 神舟十一号, 航天员 "]}
{"title": [" 新浪首页 "], "key": [" 新浪, 新浪网, SINA, sina, sina.com.cn, 新浪首页, 门户, 资讯 "]}
```

可以看到, 此时输出了两段信息, 每段包含一个网页的各项信息, 分别对应该网页的标题信息和该网页的关键词信息等, "title" 键对应的是网页的标题信息, "key" 键对应的是网页的关键词信息。

我们打开存储到的 json 文件 "D:/python35/myweb/part13/mydata1.json", 文件内容如图 14-4 所示:

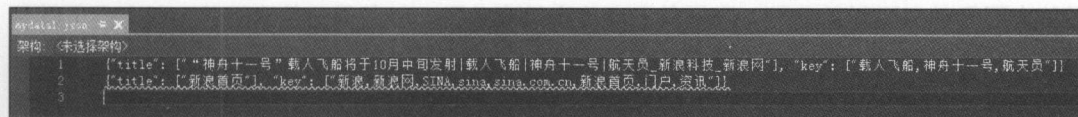


图 14-4 将多个网页的多项数据存储到 json 文件中图示

可以看到, 此时已经将多个网页中的多项含有中文的信息成功地存储到对应的 json 文件中了。接下来, 爬取下来之后, 可以存储到数据库中或者存储到某些本地的文件中。

的数据格式，属于一种轻量级的数据交换格式。

式一般为：“名称：值对”，各项数据之间可以通过逗号隔

数据一般通过 {} 括起来，里面的数据一般是键值对的形

的时候，中文信息会默认使用 ASCII 编码，所以需要把

前几章我们用 Scrapy 框架编写的爬虫项目，只能爬取起始网址中设置的网页。有时候，我们需要让爬虫持续不断地自动爬取多个网页，此时，我们需要编写自动爬取网页的爬虫。在本章中，我们以当当网为例，为大家讲解如何编写自动爬取网页的 Scrapy 爬虫，以对当当网中某个频道的所有商品的数据进行爬取。

## 15.1 实战：items 的编写

比如，我们需要爬取当当网上地方特产频道所有商品的信息（当然在实际爬取的时候，可能会有极少量数据丢失，这是正常的情况，对于大量的数据来说产生的影响微不足道），如图 15-1 所示，可以看到，这里的商品有非常多页，所以我们需要编写一个能够自动切换网页爬行的 Scrapy 爬虫来完成这个需求。

有的读者可能会思考爬取这些数据有什么实际的用处。事实上，在实际中这一类需求是非常常见的，比如我们作为第三方有时需要对当当网的商品信息进行全面的数据分析，以此挖掘出当当网上什么商品比较热销、哪些商品竞争会比较激烈、哪些关键词比较受顾客关注等一系列的信息。同理，在实际生活中我们经常还需要爬取我们关注的网站的数据信息进行数据分析，从而提取出对自己有利的信息。

由于不是这些网站的管理者，所以我们并没有这些网站的数据信息（当然有一些网站会公布一些接口来提供数据查询，但往往要全面地分析，仅仅这些接口是不够用的），所谓巧妇难为无米之炊，没有数据自然无法进行数据分析，所以此时我们可以通过爬虫将关注的数据爬取下来，爬取下来之后，可以存储到数据库中或者存储到某些本地的文件中。





图 15-1 要爬取的当当网中的网页图示

接下来，我们就为大家一步步地讲解如何编写自动爬取当当网中地方特产频道所有商品数据的爬虫。

首先创建一个名为 `autopjt` 的爬虫项目，如下所示：

```
D:\Python35\myweb\part15>scrapy startproject autopjt
New Scrapy project 'autopjt', using template directory 'd:\python35\lib\
site-packages\scrapy\templates\project', created in:
D:\Python35\myweb\part15\autopjt
You can start your first spider with:
cd autopjt
scrapy genspider example example.com
```

在创建好爬虫项目之后，首先需要编写 `items.py` 文件，在该文件中定义好我们关注的需要爬取的数据。

我们要爬取的这些网页中包含了大量的数据，但并不是每一样数据我们都关注，如果我们将这些网页所有的数据都存储起来，那么会比较乱并且浪费大量的服务器资源。比如，我们比较关注这些商品中的商品名、对应价格、对应商品链接、评论数等数据，我们可以只提取各网页中每个商品的这四项信息。

将 `autopjt` 爬虫项目下的 `items.py` 文件修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy

class AutopjtItem(scrapy.Item):
```

```
# define the fields for your item here like:
```

```
# name = scrapy.Field()
```

```
# 定义好 name 用来存储商品名
```

```
name=scrapy.Field()
```

```
# 定义好 price 用来存储商品价格
```

```
price=scrapy.Field()
```

```
# 定义好 link 用来存储商品链接
```

```
link=scrapy.Field()
```

```
# 定义好 comnum 用来存储商品评论数
```

```
comnum=scrapy.Field()
```

编写好 items.py 之后，我们就定义好了需要关注的结构化数据了。

## 15.2 实战：pipelines 的编写

编写好 items.py 文件之后，还需要对爬取到的数据做进一步的处理，比如存储到 json 文件中，此时进一步处理的操作我们可以通过编写 pipelines.py 文件实现。

可以将该爬虫项目中的 pipelines.py 文件修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-
```

```
import codecs
```

```
import json
```

```
# Define your item pipelines here
```

```
#
```

```
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
```

```
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html
```

```
class AutopjtPipeline(object):
```

```
    def __init__(self):
```

```
        # 打开 mydata.json 文件
```

```
        self.file = codecs.open("D:/python35/myweb/part15/mydata.json", "wb", encoding="utf-8")
```

```
    def process_item(self, item, spider):
```

```
        i=json.dumps(dict(item), ensure_ascii=False)
```

```
        # 每条数据后加上换行
```

```
        line = i + '\n'
```

```
        # 数据写入到 mydata.json 文件中
```

```
        self.file.write(line)
```

```
        return item
```

```
    def close_spider(self, spider):
```

```
        # 关闭 mydata.json 文件
```

```
        self.file.close()
```

此时，我们通过 pipelines.py 文件将获取到的当当网中的商品信息分别存储到了"D:/python35/myweb/part15/mydata.json" 文件中。

## 15.3 实战：settings 的编写

设置好 pipelines.py 文件之后，我们还需要编写 settings.py 文件进行相应的设置。

我们首先打开该爬虫项目中的 settings.py 文件，然后将关于 pipelines 的配置部分修改为如下所示：

```
# Configure item pipelines
# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    'autopjt.pipelines.AutopjtPipeline': 300,
}
```

因为 pipelines 默认是关闭的，所以我们需要通过上面的设置将 pipelines 开启，这样，在 15.2 节中设置的 pipelines.py 文件才会生效。

为了避免当网服务器通过我们的 Cookie 信息识别我们的爬虫行为，从而对我们进行屏蔽，所以接下来，我们还需要关闭本地的 Cookie。

可以通过修改该爬虫项目中的 settings.py 文件中关于 Cookie 部分的设置实现关闭 Cookie 的操作。

settings.py 文件中关于 Cookie 部分的设置如下所示：

```
# Disable cookies (enabled by default)
#COOKIES_ENABLED = False
```

可以看到，此时对应的设置处于被注释的状态，此时默认是开启 COOKIES 的。所以我们需要将上面的代码修改为如下所示：

```
# Disable cookies (enabled by default)
COOKIES_ENABLED = False
```

我们解除了 COOKIES\_ENABLED = False 部分的注释，将 COOKIES\_ENABLED 的值设置为 False，即通过该项设置可以不使用 Cookie。这样，对方的服务器就无法根据 Cookie 信息对我们进行识别及屏蔽了，当然如果对方服务器通过其他方面对我们进行屏蔽，我们可以依据本书第 12 章中的对应策略进行相应的反屏蔽处理。

## 15.4 自动爬虫编写实战

设置好 settings.py 文件之后，我们需要对该项目中最核心的部分——爬虫文件部分进行相应的编写，来实现目标网页的自动爬取以及关键信息的提取。

首先，需要在该爬虫项目对应的目录下创建一个爬虫文件，如下所示：

```
D:\Python35\myweb\part15>cd .\autopjt\
D:\Python35\myweb\part15\autopjt>scrapy genspider -t basic autopsp dangdang.com
```

```
Created spider 'autospd' using template 'basic' in module:
Autopjt.spiders.autospd
```

在上面的代码中，我们在该爬虫项目中创建了一个名为 autospd 的爬虫文件，依据的爬虫模板是 basic。

为了实现网页的自动爬行，我们需要对要爬行网页的 URL 地址进行观察，发现其中的规律。

打开当当网地方特产频道中的第 1 页，网址如下所示：

```
http://category.dangdang.com/cid4002203.html.75
```

然后点击页面中的“下一页”按钮，此时界面切换到了当当网地方特产频道中商品列表的第 2 页，网址变为：

```
http://category.dangdang.com/pg2-cid4002203.html
```

然后再点击“下一页”按钮，此时界面切换到了当当网地方特产频道中商品列表的第 3 页，网址变化为：

```
http://category.dangdang.com/pg3-cid4002203.html
```

我们可以发现其中的规律，即网址的格式为：

```
http://category.dangdang.com/pg[ 第几页 ]-cid4002203.html
```

但是第 1 页网址为 <http://category.dangdang.com/cid4002203.html> 不符合该格式，那么我们猜测，是否第 1 页的网址可以按格式写为 <http://category.dangdang.com/pg1-cid4002203.html> 呢？

我们将网址 <http://category.dangdang.com/pg1-cid4002203.html> 复制到浏览器中并访问，发现该网址与网址 <http://category.dangdang.com/cid4002203.html> 是同一个页面，说明第 1 页也是满足格式 [http://category.dangdang.com/pg\[ 第几页 \]-cid4002203.html](http://category.dangdang.com/pg[ 第几页 ]-cid4002203.html) 的。

总结出该规律后，我们在爬虫文件中就可以通过 for 循环将所有的页面都自动爬取下来。

接下来我们还需要分析如何信息提取。

因为我们只关注各网页中的商品名称、商品价格、商品链接、商品评论数的信息，所以我们需要将这些关注的信息从大量的数据中提取出来，提取的方法有很多种，在这里我们使用 XPath 表达式进行提取。

首先，我们可以看到网页中的信息，如图 15-2 所示：

我们需要将图 15-2 中所有商品的商品名、价格、链接、商品评论数的数据从网页中提取出来，所以我们需要查看该网页的源码进行相应的分析。

在页面中单击右键，然后查看源代码，我们会发现出现的大量的代码。

为了更好的定位分析我们关注的信息，可以看到图 15-2 中第一个商品是“[ 当当自营 ] 鲁花玉米油桶装 5L”，所以我们可以根据该商品名定位出源代码中对应的位置，可以在源代



码界面中使用 Ctrl+F 键查找对应关键词：“鲁花玉米油”，然后可以找到如图 15-3 所示的信息：



图 15-2 要爬取的第一网页的部分商品信息

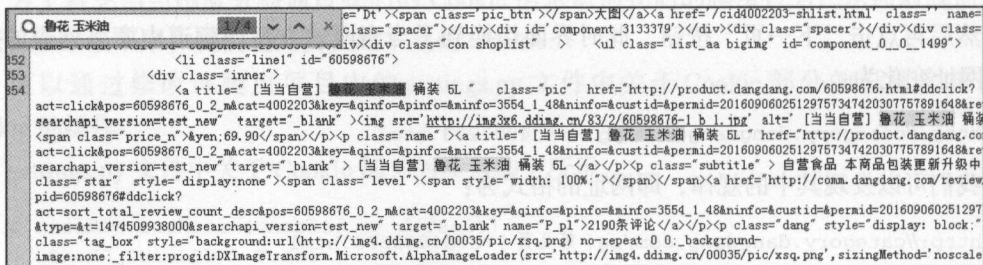


图 15-3 要爬取的第一网页对应源码

图 15-3 中的关于“鲁花玉米油”的代码与图 15-2 中的“[ 当当自营 ] 鲁花玉米油桶装 5L”商品对应。

为了方便大家分析，我们将图 15-3 中的相关代码复制出来，如下所示：

```
<div class="inner">
<a title=" [ 当当自营 ] 鲁花玉米油桶装 5L " class="pic" href="http://product.dangdang.
com/60598676.html#ddclick?act=click&pos=60598676_0_2_m&cat=4002203&key=&qinfo=&pin
fo=&minfo=3554_1_48&ninfo=&custid=&permid=20160906025129757347420307757891648&ref=
&rcount=&type=&t=1474509938000&searchapi_version=test_new" target="_blank" ><img
src='http://img3x6.ddimg.cn/83/2/60598676-1_b_1.jpg' alt=' [ 当当自营 ] 鲁花玉米油桶装 5L
' /></a><p class="price" ><span class="price_n">¥69.90</span></p><p class="name"
><a title=" [ 当当自营 ] 鲁花玉米油桶装 5L " href="http://product.dangdang.com/60598676.
html#ddclick?act=click&pos=60598676_0_2_m&cat=4002203&key=&qinfo=&pinfo=&minfo=3
554_1_48&ninfo=&custid=&permid=20160906025129757347420307757891648&ref=&rcount=&
type=&t=1474509938000&searchapi_version=test_new" target="_blank" > [ 当当自营 ] 鲁
花玉米油桶装 5L </a></p><p class="subtitle" > 自营食品 本商品包装更新升级中新老包装随机发放
</p><p class="star" style="display:none"><span class="level"><span style="width:
100%; "></span></span><a href="http://comm.dangdang.com/review/reviewlist.php?pid=60
598676#ddclick?act=sort_total_review_count_desc&pos=60598676_0_2_m&cat=4002203&key
=&qinfo=&pinfo=&minfo=3554_1_48&ninfo=&custid=&permid=201609060251297573474203077
57891648&ref=&rcount=&type=&t=1474509938000&searchapi_version=test_new" target="_
```

```
blank" name="P_pl">2190 条 评论</a></p><p class="dang" style="display: block;">当
当 自 营</p><span class="tag_box" style="background:url(http://img4.ddimg.cn/00035/
pic/xsq.png) no-repeat 0 0;_background-image:none;_filter:progid:DXImageTransform.
Microsoft.AlphaImageLoader(src='http://img4.ddimg.cn/00035/pic/xsq.png',sizingMethod
='noscale');"></span></div>
```

从上面代码中可以归纳总结出提取我们关注的信息的 XPath 表达式。

首先,我们归纳分析提取商品名的 XPath 表达式。

我们知道,在网页源码中,商品名对应的源码部分为:

```
<a title=" [ 当当自营 ] 鲁花玉米油桶装 5L " class="pic" ...
```

所以,可以得到 XPath 表达式为:

```
"//a[@class='pic']/@title"
```

表达式中提取网页中所有的 class 属性为 pic 的 a 标签中的 title 属性对应的值。

接下来,我们需要归纳分析提取商品价格信息的 XPath 表达式。

在网页源码中,商品价格对应的网页源码部分为:

```
<p class="price" ><span class="price_n">&yen;69.90</span>...
```

可以看到,此时商品价格在 span 标签内,并且此时的 span 标签具有一个特点,就是其 class 的属性值为 price\_n,所以,我们可以得到提取商品价格信息的 XPath 表达式为:

```
"//span[@class='price_n']"
```

随后,我们还需要提取商品对应的链接信息。

在网页源码中,商品链接信息对应的源码部分为:

```
...<a title=" [ 当当自营 ] 鲁花玉米油桶装 5L " class="pic" href="http://product.
dangdang.com/60598676.html#ddclick?act=click&pos=60598676_0_2_m&cat=4002203&key=&qinfo=
&pinfo=&minfo=3554_1_48&ninfo=&custid=&permid=20160906025129757347420307757891648
&ref=&rcount=&type=&t=1474509938000&searchapi_version=test_new" target="_blank" >...
```

可以看得到,在上面的代码中,商品的链接为:

```
"http://product.dangdang.com/60598676.html#ddclick?act=click&pos=60598676_0_2_m&cat=
4002203&key=&qinfo=&pinfo=&minfo=3554_1_48&ninfo=&custid=&permid=2016090602512975
7347420307757891648&ref=&rcount=&type=&t=1474509938000&searchapi_version=test_new"
```

该链接在源码中的 a 标签下,并且此时的 a 标签具有一个特征,就是其 class 属性为 pic,对应链接为该 a 标签下的 href 属性的值。

所以可以得到,提取商品链接信息的 XPath 表达式为:

```
"//a[@class='pic']/@href"
```

接下来我们还需要提取商品的评论数,实际中商品的评论数在我们的数据分析过程具

有非常大的参考意义，比如可以依据商品评论数可以在一定程度上分析对应商品的热销程度等。那么商品的评论数信息应该如何提取呢？

同样，我们可以从商品评论数对应的网页源码中进行相应分析，在上面的网页源码中商品评论数对应的源码部分为：

```
...
<a href="http://comm.dangdang.com/review/reviewlist.php?pid=60598676#ddclick
?act=sort_total_review_count_desc&pos=60598676_0_2_m&cat=4002203&key=&qinfo=
&pinfo=&minfo=3554_1_48&ninfo=&custid=&permid=20160906025129757347420307757
891648&ref=&rcount=&type=&t=1474509938000&searchapi_version=test_new" target="_blank"
name="P_pl">2190 条评论 </a></p>
...
```

此时，上面代码的评论数部分的数据为“2190 条评论”，可以看到，商品评论数在 a 标签下，并且此时的 a 标签有一个特点，就是其 name 属性的值为 P\_pl，所以此时我们可以得到提取商品评论数信息的 XPath 表达式为：

```
 "//a[@name='P_pl']/text()"
```

通过上面的分析，我们已经分别得到了提取商品名称、商品价格、商品链接、商品评论数的 XPath 表达式。

接下来，我们可以编写该爬虫项目中的爬虫文件 autospd.py，我们可以将爬虫文件代码修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-
import scrapy
from autopjt.items import AutopjtItem
from scrapy.http import Request

class AutospdSpider(scrapy.Spider):
    name = "autospd"
    allowed_domains = ["dangdang.com"]
    start_urls = (
        'http://category.dangdang.com/pg1-cid4002203.html',
    )

    def parse(self, response):
        item=AutopjtItem()
        # 通过各 XPath 表达式分别提取商品的名称、价格、链接、评论数等信息
        item["name"]=response.xpath("//a[@class='pic']/@title").extract()
        item["price"]=response.xpath("//span[@class='price_n']/text()").extract()
        item["link"]=response.xpath("//a[@class='pic']/@href").extract()
        item["comnum"]=response.xpath("//a[@name='P_pl']/text()").extract()
        # 提取完后返回 item
        yield item
        # 接下来很关键，通过循环自动爬取 75 页的数据
        for i in range(1,76):
            # 通过上面总结的网址格式构造要爬取的网址
```

```
url="http://category.dangdang.com/pg"+str(i)+"-cid4002203.html"
# 通过 yield 返回 Request, 并指定要爬取的网址和回调函数
# 实现自动爬取
yield Request(url, callback=self.parse)
```

上面的爬虫文件中, 关键部分为通过 XPath 表达式对我们所关注的数据进行提取、要爬取的网址的构造以及通过 yield 返回 Request 实现网页的自动爬取等。

如果我们要编写其他自动爬取网页的爬虫, 也可以按照类似的方法实现, 不同的地方只是提取的信息及表达式不同、网址的构造不同而已。

## 15.5 调试与运行

接下来我们可以进行相应的运行与调试。

我们在使用 scrapy crawl 指令运行对应爬虫的时候, 可能会爬取失败, 并出现以下提示:

```
2016-09-22 16:15:00 [scrapy] DEBUG: Forbidden by robots.txt: <GET http://category.
dangdang.com/pg1-cid4002203.html>
2016-09-22 16:15:00 [scrapy] INFO: Closing spider (finished)
```

可以看到, 提示中说当当网的 robots.txt 文件禁止我们爬取。我们之前也为大家初步地提过 robots.txt 文件。该文件是爬虫协议, 一般情况下, 我们都应该遵守该协议。但如果我们想对该网站进行爬取, 则需要修改一下该爬虫项目中的设置文件 settings.py, 让爬虫项目不遵循 robots.txt 规则即可。

打开该爬虫项目 autopjt 下的 settings.py 文件, 找到 ROBOTSTXT 部分的设置, 默认的设置如下所示:

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = True
```

可以看到, 默认 ROBOTSTXT\_OBEY 的值为 True, 即默认的情况是遵守 robots.txt 规则的, 我们可以修改为:

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = False
```

设置后, 该爬虫项目就可以不遵循 robots.txt 规则进行爬取了, 此时就不会出现“Forbidden by robots.txt”之类的错误了。

解决了该问题之后, 可以再次运行 autopjt 爬虫项目下的 autospd 爬虫文件, 如下所示:

```
D:\Python35\myweb\part15\autopjt>scrapy crawl autospd --nolog
```

运行后, 会通过 pipelines.py 文件中编写的代码, 将爬取到的对应的数据存储到本地文



件 "D:/python35/myweb/part15/mydata.json" 中, 我们可以查看对应的 json 文件 (由于数据太多, 截图中显示不完整, 为了方便读者学习, 笔者将对应的文件传到了服务器中, 读者可以通过网址 <http://iqianyue.com/weisuenbook/pyspd/part15/mydata.json> 下载该 json 文件, 需要下载到本地打开, 在网页直接浏览可能会出现编码问题), json 文件中的内容如图 15-4 所示。

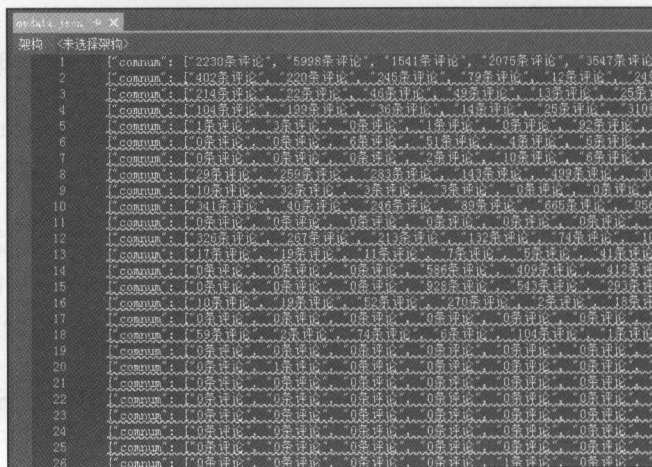


图 15-4 mydata.json 文件中的部分信息

可以看得, 此时已经将对应爬取到的数据存储到了 json 文件中。

此时的存储形式如下所示:

```
第 1 页的数据 { "comnum": [第 1 个商品评论数, 第 2 个, ...], "link": ["第 1 个商品链接", "第 2 个商品链接", ...], ..... }
第 2 页的数据 { "comnum": [第 1 个商品评论数, 第 2 个, ...], "link": ["第 1 个商品链接", "第 2 个商品链接", ...], ..... }
.....
第 50 页的数据 { "comnum": [第 1 个商品评论数, 第 2 个, ...], "link": ["第 1 个商品链接", "第 2 个商品链接", ...], ..... }
```

可以看到, 此时每一行存储的是每一页商品的数据, 而每一页中会有多个商品, comnum 键中存储的是每一页中所有商品的评论数信息, comnum 键对应的值是一个列表, 列表中分别存储当前页中各商品的评论数, 每个元素对应 1 个商品。同样, price 键中存储的是每一页中所有商品的价格信息, link 键中存储的是每一页中所有商品的链接信息, name 键中存储的是每一页中所有商品的名称信息。

如果我们觉得太乱, 想存储为如下形式:

```
{ "comnum": "第 1 个商品评论数", "price": "第 1 个商品价格", "link": "第 1 个商品链接", "name": "第 1 个商品名" }
.....
{ "comnum": "第 n 个商品评论数", "price": "第 n 个商品价格", "link": "第 n 个商品链接", "name": "第 n 个商品名" }
```

即如果我们想每一行存储一个商品的对应信息,希望 comnum、price、link、name 等键的值只对应一个商品的信息,也是可以的,只不过需要处理一下。

对数据的进一步处理一般可以通过 pipelines.py 文件进行,所以此时我们可以修改该项目中的 pipelines.py 文件,修改为如下所示,关键部分已给出注释:

```
# -*- coding: utf-8 -*-
import codecs
import json

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

class AutopjtPipeline(object):
    def __init__(self):
        # 此时存储到的文件是 mydata2.json, 不与之前存储的文件 mydata.json 冲突
        self.file = codecs.open("D:/python35/myweb/part15/mydata2.json", "wb", encoding="utf-8")

    def process_item(self, item, spider):
        # item=dict(item)
        # print(len(item["name"]))
        # 每一页中包含多个商品信息, 所以可以通过循环, 每一次处理一个商品
        # 其中 len(item["name"]) 为当前页中商品的总数, 依次遍历
        for j in range(0, len(item["name"])):
            # 将当前页的第 j 个商品的名称赋值给变量 name
            name=item["name"][j]
            price=item["price"][j]
            comnum=item["comnum"][j]
            link=item["link"][j]
            # 将当前页下第 j 个商品的 name、price、comnum、link 等信息处理一下
            # 重新组合成一个字典
            goods={"name":name,"price":price,"comnum":comnum,"link":link}
            # 将组合后的当前页中第 j 个商品的数据写入 json 文件
            i=json.dumps(dict(goods), ensure_ascii=False)
            line = i + '\n'
            self.file.write(line)

        # 返回 item
        return item

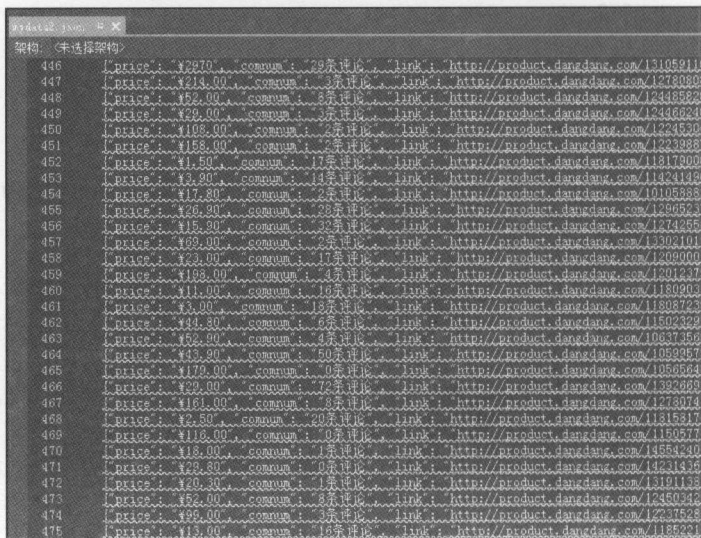
    def close_spider(self, spider):
        self.file.close()
```

编写好 pipelines.py 文件进行进一步处理之后,我们可以重新运行一下该爬虫,如下所示:

```
D:\Python35\myweb\part15\autopjt>scrapy crawl autospd --nolog
```

运行完成之后,我们可以打开存储的对应的 JSON 文件 "D:/python35/myweb/part15/mydata2.

json", 结果如图 15-5 所示 (由于数据太多, 截图中显示不完整, 为了方便读者学习, 笔者将对应的文件传到了服务器中, 读者可以通过网址 <http://iqianyue.com/weisuenbook/pyspd/part15/mydata2.json> 下载该 json 文件, 需要下载到本地打开, 在网页直接浏览可能会出现编码问题)。



```

446 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/13105911", "name": "127.00 22条评论"}
447 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
448 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
449 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
450 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
451 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
452 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
453 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
454 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
455 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
456 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
457 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
458 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
459 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
460 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
461 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
462 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
463 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
464 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
465 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
466 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
467 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
468 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
469 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
470 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
471 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
472 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
473 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
474 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}
475 {"price": 127.00, "comnum": 22, "link": "http://product.dandant.com/1270499", "name": "127.00 22条评论"}

```

图 15-5 mydata2.json 文件中的部分信息

可以看到, 此时每一行中存储的是一个商品的信息, 分别包含该商品的 price (价格)、comnum (评论数)、link (链接)、name (商品名) 等信息。

## 15.6 小结

- 1) 有时候, 我们需要让爬虫持续不断地自动爬取多个网页, 此时, 我们需要编写自动爬取网页的爬虫。
- 2) 在实际中我们经常需要爬取我们关注网站的数据信息进行数据分析, 从而提取出对自己有用的信息。
- 3) 为了避免对方服务器通过我们的 Cookie 信息识别我们的爬虫行为从而对我们进行屏蔽, 所以我们通常需要关闭本地的 Cookie, 可以通过 COOKIES\_ENABLED 进行设置。
- 4) Scrapy 项目默认会遵守 robots 协议进行爬取, 若遇到限制, 可以通过设置 ROBOTSTXT\_OBEY 解决, 当然我们需要在不违法且遵守道德的情况下使用。

## CrawlSpider

在上一章，我们学习了如何编写自动爬取网页的爬虫，我们通过循环实现各个网页的依次遍历爬取。其实，在 Scrapy 框架中，提供了一种自带的自动爬取网页的爬虫 CrawlSpider，我们可以使用 CrawlSpider 轻松实现网页的自动爬取，CrawlSpider 与上一章中我们编写的自动爬取网页的爬虫实现机制有所不同，但可以得到类似的结果，具体实现机制有何不同在学完本章之后我们也会有所了解。

在本章中，我们会讲解 CrawlSpider 的详细使用方法。

### 16.1 初识 CrawlSpider

接下来我们来初步认识一下 CrawlSpider。

如下所示，我们创建了一个名为 mycwpjt 的爬虫项目。

```
D:\Python35\myweb\part16>scrapy startproject mycwpjt
New Scrapy project 'mycwpjt', using template directory 'd:\python35\lib\
site-packages\scrapy\templates\project', created in:
D:\Python35\myweb\part16\mycwpjt
You can start your first spider with:
cd mycwpjt
scrapy genspider example example.com
```

创建爬虫项目之后，我们可以进入该爬虫项目，并通过 `scrapy genspider -l` 查看该爬虫项目下拥有的爬虫模板，如下所示：

```
D:\Python35\myweb\part16\mycwpjt>scrapy genspider -l
```



Available templates:

```
basic
crawl
csvfeed
xmlfeed
```

可以看到，在爬虫模板中有一个模板名为 `crawl`，该模板就是 `CrawlSpider` 爬虫的模板，我们如果要创建一个 `CrawlSpider` 爬虫，可以依据 `crawl` 爬虫模板创建即可，如下所示，我们依据 `crawl` 爬虫模板创建了一个名为 `weisuen` 的 `CrawlSpider` 爬虫。

```
D:\Python35\myweb\part16\mycwpjt>scrapy genspider -t crawl weisuen sohu.com
Created spider 'weisuen' using template 'crawl' in module:
Mycwpjt.spiders.weisuen
```

然后我们可以通过编辑器打开该爬虫文件 `weisuen.py`，可以看到，爬虫文件里的默认内容为：

```
# -*- coding: utf-8 -*-
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule

from mycwpjt.items import MycwpjtItem

class WeisuenSpider(CrawlSpider):
    name = 'weisuen'
    allowed_domains = ['sohu.com']
    start_urls = ['http://www.sohu.com/']

    rules = (
        Rule(LinkExtractor(allow=r'Items/'), callback='parse_item', follow=True),
    )

    def parse_item(self, response):
        i = MycwpjtItem()
        #i['domain_id'] = response.xpath('//input[@id="sid"]/@value').extract()
        #i['name'] = response.xpath('//div[@id="name"]').extract()
        #i['description'] = response.xpath('//div[@id="description"]').extract()
        return i
```

这就是一个 `CrawlSpider` 爬虫的默认内容，在上面的代码中，`start_urls` 属性一般设置要爬取的起始网址，`rules` 一般设置自动爬行的规则，`LinkExtractor` 为链接提取器，一般可以用来提取页面中满足条件的链接，以供下一次爬行使用，`parse_item` 方法主要用于编写爬虫的处理过程。

## 16.2 链接提取器

在上面代码 `rules` 部分中的 `LinkExtractor` 为链接提取器，如下所示：

```
rules = (
    Rule(LinkExtractor(allow=r'Items/'), callback='parse_item', follow=True),
)
```

链接提取器主要负责将 response 响应中符合条件的链接提取出来，这些条件我们可以自行设置，常见的设置参数如表 16-1 所示：

表 16-1 LinkExtractor 中对应的参数及含义

参数名	参数含义
allow	提取符合对应正则表达式的链接
deny	不提取符合对应正则表达式的链接
restrict_xpaths	使用 XPath 表达式与 allow 共同作用提取出同时符合对应 XPath 表达式和对应正则表达式的链接
allow_domains	允许提取的域名，比如我们想只提取某个域名下的链接时会用到
deny_domains	禁止提取的域名，比如我们需要限制一定不提取某个域名下的链接时会用的

比如，我们想提取链接中有“.shtml”字符串的链接，可以将 rules 规则设置为如下所示：

```
rules = (
    Rule(LinkExtractor(allow=('.shtml')), callback='parse_item', follow=True),
)
```

比如，我们想进一步限制只能提取搜狐官方的链接（sohu.com），可以将域名设置为只允许提取 sohu.com 域名的链接，将 rules 设置为如下所示：

```
rules = (
    Rule(LinkExtractor(allow=('.shtml'), allow_domains=(sohu.com)), callback='parse_item',
        follow=True),
)
```

设置好之后，就会按照对应的规则提取 response 响应中符合条件的链接，提取出来这些链接之后会进一步爬取这些链接。

## 16.3 实战：CrawlSpider 实例

接下来我们实战讲解一下 CrawlSpider。

首先，我们需要知道 CrawlSpider 的工作流程，CrawlSpider 的主要工作流程如图 16-1 所示：

可以看得到，CrawlSpider 爬虫会根据链接提取器中设置的规则自动的提取符合条件的网页链接，提取之后再自动的对这些链接进行爬行，形成一个循环，如果链接设置为跟进，则会一直循环下去，如果链接设置为不跟进，则第一次循环后就会断开，链接是否跟进可以通过 rules 中的 follow 参数设置，follow 参数的值为 True 表示跟进，follow 参数的值为 False 表示不跟进，基于爬虫模板创建 CrawlSpider 后，默认情况为跟进链接。

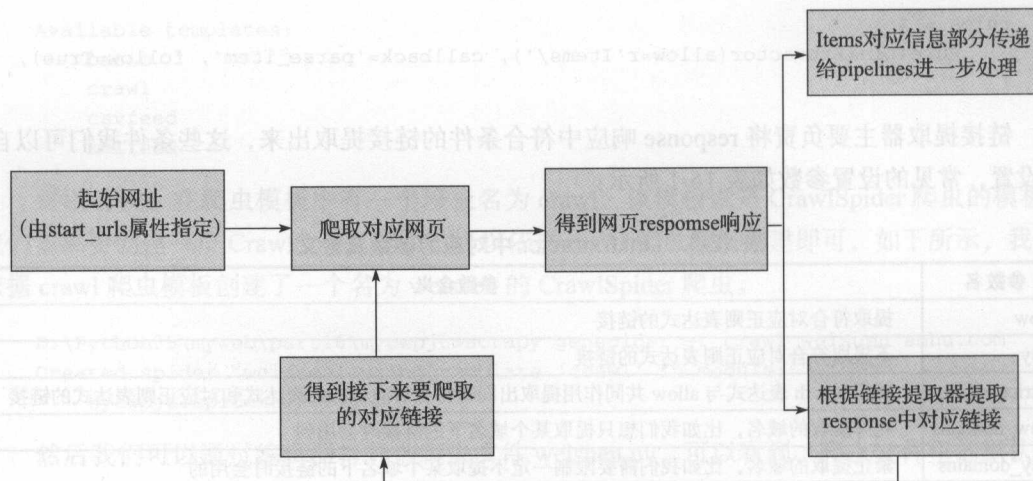


图 16-1 CrawlSpider 的主要工作流程

假如我们要自动爬取搜狐新闻网站中的新闻，可以通过 CrawlSpider 爬虫实现，具体实现过程如下所示。

首先，需要创建一个爬虫项目，这里我们的爬虫项目可以使用 16.1 节中创建的爬虫项目 mycwpjt，然后我们需要编写爬虫项目中的 items.py 文件，假如我们想提取新闻的标题和新闻的链接，可以将 items.py 文件修改为如下所示：

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy

class MycwpjtItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    name=scrapy.Field()
    link=scrapy.Field()
```

此时，我们可以用 name 存储新闻的标题，用 link 存储对应新闻的链接。

然后我们可以编写爬虫项目中的 pipelines.py 文件，假如我们需要直接输出提取出来的对应新闻的标题和链接，可以将 pipelines.py 文件修改为：

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
```

```
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html
```

```
class MycwpjtPipeline(object):
    def process_item(self, item, spider):
        print(item["name"])
        print(item["link"])
        print("-----")
        return item
```

首先，我们输出新闻的标题，然后输出新闻对应的链接，随后再输出 "-----" 作为不同新闻之间的分隔符。

接下来，我们还需要编写爬虫项目中的设置文件 `settings.py`，在设置文件中，我们需要开启对应的 `ITEM_PIPELINES`，在本项目中，可以将设置文件中关于 `ITEM_PIPELINES` 的部分设置为：

```
# Configure item pipelines
# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    'mycwpjt.pipelines.MycwpjtPipeline': 300,
}
```

配置好项目的设置文件 `settings.py` 之后，我们还需要编写本项目中的核心部分——爬虫文件 `weisuen.py`，可以将爬虫文件 `weisuen.py` 修改为如下所示，关键部分已给出注释。

```
# -*- coding: utf-8 -*-
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule

from mycwpjt.items import MycwpjtItem

class WeisuenSpider(CrawlSpider):
    name = 'weisuen'
    allowed_domains = ['sohu.com']
    start_urls = ['http://news.sohu.com/']

    rules = (
        # 新闻网页的 URL 地址类似于：
        # "http://news.sohu.com/20160926/n469167364.shtml"
        # 所以可以得到提取的正则表达式为 '.*/n.*?shtml'
        Rule(LinkExtractor(allow=('.*/n.*?shtml'), allow_domains=('sohu.com')),
            callback='parse_item', follow=True),
    )

    def parse_item(self, response):
```



```

i = MycwpjtItem()
# 根据 XPath 表达式提取新闻网页中的标题
i["name"]=response.xpath("/html/head/title/text()").extract()
# 根据 XPath 表达式提取当前新闻网页的链接
i["link"]=response.xpath("//link[@rel='canonical']/@href").extract()
return i

```

上面代码中值得注意的是，要提取当前新闻网页对应的链接，我们可以观察新闻网页的源码，该网页对应链接的部分源码为如下形式（实践中会发现多年以前的一些搜狐新闻对应链接部分的源代码不具备该规律，所以通过该规律无法提取多年以前的搜狐新闻的链接，如果要提取这些链接，可以通过其他 XPath 表达式进行）：

```
<link rel="canonical" href="http://news.sohu.com/20160926/n469156749.shtml"/>
```

所以可以归纳总结得到提取当前新闻网页链接的 XPath 表达式为：

```
//link[@rel='canonical']/@href"
```

编写好爬虫文件 weisuen.py 之后，接下来我们可以运行该爬虫，如下所示：

```
D:\Python35\myweb\part16\mycwpjt>scrapy crawl weisuen --nolog
```

```
['  迈克尔拥抱小布什引热议媒体笑称“抱错总统”（图）- 搜狐新闻 ']
```

```
['http://news.sohu.com/20160926/n469162020.shtml']
```

```
['  韩媒：朝外相联大期间无显著活动成果或被美限制 - 搜狐新闻 ']
```

```
['http://news.sohu.com/20160926/n469159506.shtml']
```

```
['  瑞典南部城市发生枪击案导致 4 人受伤枪手在逃 - 搜狐新闻 ']
```

```
['http://news.sohu.com/20160926/n469169602.shtml']
```

```
['  日本冲绳本岛近海发生 5.7 级地震未引发海啸 - 搜狐新闻 ']
```

```
['http://news.sohu.com/20160926/n469183662.shtml']
```

```
['  墨西哥湾一艘油轮失火火势剧烈冒出浓密黑烟 - 搜狐新闻 ']
```

```
['http://news.sohu.com/20160926/n469183709.shtml']
```

```
['  特朗普：若当选将承认耶路撒冷为以色列首都 - 搜狐新闻 ']
```

```
['http://news.sohu.com/20160926/n469183764.shtml']
```

```
...由于获取的结果太多，为了方便读者阅读，在此省略部分代码...
```

```
['iPhone7 嘶嘶声无解不过新功能来袭算是安慰 - 搜狐科技 ']
```

```
['http://it.sohu.com/20160922/n468991170.shtml']
```

此时，会进行链接的跟进，所以会一直根据网站中的链接一直爬取下去，如果我们想在爬行的过程中停止，可以按 Ctrl+C 终止爬行。

如果我们不想进行链接的跟进，可以将 rules 部分设置为：

```
rules = (
```

```
Rule(LinkExtractor(allow=('.*?/n.*?shtml'),allow_domains=('sohu.com')), call
back='parse_item', follow=False),
)
```

设置好后，可以再次运行爬虫，发现此时爬虫就不进行链接跟进了，很快（大约十几秒）就将爬取任务运行完了。

同样，我们还可以在链接提取器 LinkExtractor 中通过 restrict\_xpaths 参数设置一个 XPath 表达式与 allow 参数中设置的正则表达式共同来实现链接的过滤与提取。

此时，相信大家已经能够用 CrawlSpider 爬虫来实现一些需求了，接下来我们为大家总结一下使用 CrawlSpider 和第 15 章中使用的自动爬取网页爬虫的主要区别：

1) CrawlSpider 与第 15 章中学习的自动爬虫实现原理不同，CrawlSpider 主要是根据网页页面之间的链接关系依次自动爬行，第 15 章中学习的自动爬虫主要依据 URL 链接之间的规律使用循环（比如 for 循环）进行自动爬取。

2) 由于实现原理不同，第 15 章中学习的自动爬取网页爬虫适合爬取多个有规律的页面，因为使用 for 循环来遍历多个 URL 有规律的网页比较方便。

3) 由于实践原理不同，使用 CrawlSpider 爬虫可以适应无规律的 URL 网址自动爬取任务，因为 CrawlSpider 爬虫是根据页面中的链接进行自动爬取的，当然 CrawlSpider 爬虫也可以爬取有规律的 URL 网址对应的网页。

4) 在一定的程度上，如果要做通用爬虫，CrawlSpider 会更适合，而使用第 15 章中的自动爬虫会略显不足。

## 16.4 小结

1) 在 Scrapy 框架中，提供了一种自带的自动爬取网页的爬虫 CrawlSpider，我们可以使用 CrawlSpider 轻松实现网页的自动爬取。

2) CrawlSpider 爬虫会根据链接提取器中设置的规则自动的提取符合条件的网页链接，提取之后再自动的对这些链接进行爬行，形成一个循环，如果链接设置为跟进，则会一直循环下去，如果链接设置为不跟进，则第一次循环后就会断开，链接是否跟进可以通过 rules 中的 follow 参数设置。

3) CrawlSpider 与第 15 章中学习的自动爬虫实现原理不同，CrawlSpider 主要是根据网页页面之间的链接关系依次自动爬行，第 15 章中学习的自动爬虫主要依据 URL 链接之间的规律使用循环（比如 for 循环）进行自动爬取。

## Scrapy 高级应用

通过前几章的学习，我们已经能够使用 Scrapy 框架编写出一些常见的网络爬虫，在本章中，我们将接触 Scrapy 框架的一些高级应用，比如将爬取到的数据写入 MySQL 数据库中等。

### 17.1 如何在 Python3 中操作数据库

要想在 Scrapy 爬虫项目中将爬取到的内容写进数据库中，就得首先学会如何在 Python3 中操作数据库，数据库有很多种，在此我们会以 MySQL 数据库为例进行实际讲解。

在 Python2.X 中操作 MySQL 数据库可以通过 Python 的 MySQLdb 模块实现，由于目前 Python3.5 无法支持 MySQLdb 模块，所以此时我们可以通过其他方式来实现在 Python 中操作 MySQL 数据库。

在 Python3.X 中我们可以使用 pymysql 模块来通过 Python 代码操作 MySQL 数据库。首先，需要在电脑中安装 pymysql 模块，我们可以使用 `pip install pymysql` 进行安装，如下所示：

```
D:\Python35\myweb\part16\mycwpjt>pip install pymysql
Collecting pymysql
  Downloading PyMySQL3-0.5.tar.gz
    running egg_info
    .....
    Stored in directory: C:\Users\Administrator.USER-20160828PN\AppData\Local\
    pip\Cache\wheels\bf\84\b3\c2cb0d3d8e99f408976e112f65ba4780cbfb446a606dd620db
    Successfully built pymysql
    Installing collected packages: pymysql
    Successfully installed pymysql-0.5
```

执行之后, 可以看到 “Successfully installed pymysql3-0.5” 这样的信息, 说明此时 pymysql 模块已经成功安装。

为了方便在本地能够进行测试, 我们需要在本地搭建好 MySQL 数据库服务器。搭建的方式有很多, 比如可以使用 phpStudy 集成工具来安装 MySQL 服务器, phpStudy 集成工具可以直接在互联网中进行搜索并下载。

安装好 phpStudy 集成工具之后, 我们可以打开该软件, 会出现如图 17-1 所示界面:

此时点击 “启动” 或 “重启” 按钮即可开启 MySQL 服务器, 开启后界面如图 17-2 所示。

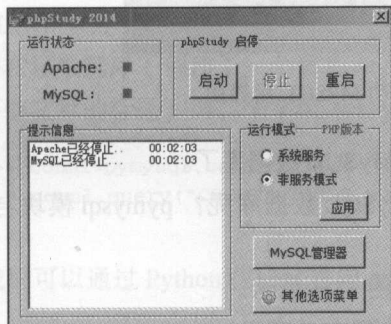


图 17-1 打开 phpStudy 集成工具后图示

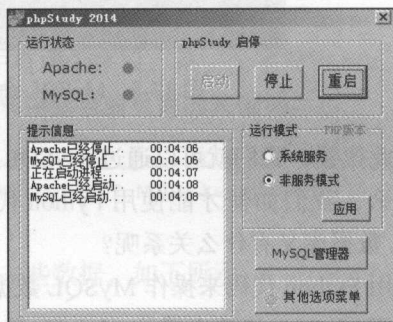


图 17-2 启动 MySQL 服务器后图示

然后我们可以试着通过终端连接 MySQL 服务器。

单击界面中的 “其他选项菜单” 按钮, 然后选择 “MySQL 工具 --MySQL 命令行”, 如图 17-3 所示。

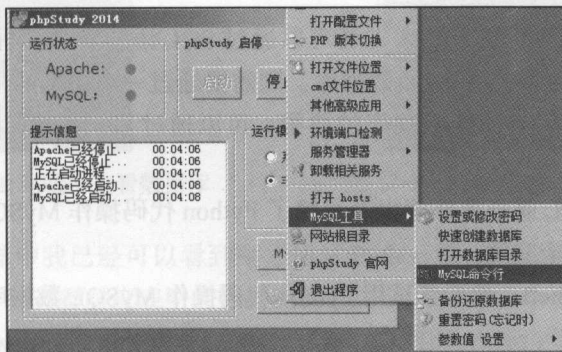
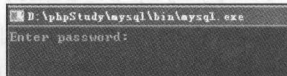


图 17-3 打开 MySQL 命令行操作图示

单击 “MySQL 命令行” 之后, 会调出 MySQL 命令行连接界面, 如图 17-4 所示。

phpStudy 集成工具中的 MySQL 服务器默认账号为 root, 默认密码为 root, 此时, 我们需要输入密码 (password), 图 17-4 MySQL 命令行连接界面





输入 root，然后按回车键，出现如图 17-5 所示界面。

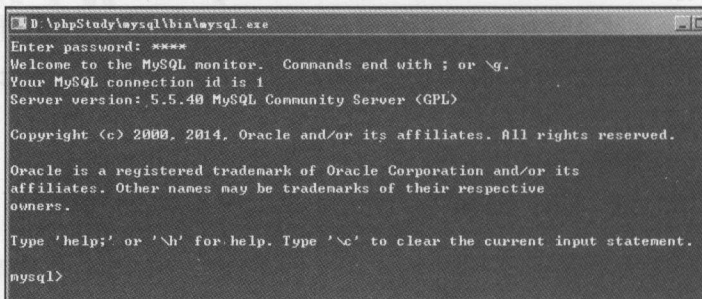


图 17-5 MySQL 命令行中连接成功 MySQL 后界面

在此位置，我们就可以通过 MySQL 终端来操作 MySQL 数据库了。

那么，我们如何才能使用 Python 代码来操作 MySQL 数据库呢？pymysql 模块与操作 MySQL 数据库又有什么关系呢？

使用 Python 代码来操作 MySQL 数据库的过程如图 17-6 所示。

使用 MySQL 终端来操作 MySQL 数据库的过程如图 17-7 所示。

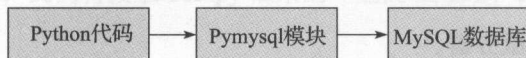


图 17-6 使用 Python 代码来操作 MySQL 数据库的过程图示

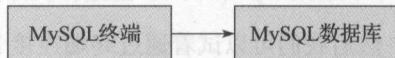


图 17-7 使用 MySQL 终端来操作 MySQL 数据库的过程图示

可以看得到，使用 MySQL 终端工具可以通过 SQL 语句直接操作 MySQL 数据库，而如果要使用 Python 代码操作 MySQL 数据库，则需要通过一个中间的桥梁进行，pymysql 模块就是 Python 代码操作 MySQL 数据库的桥梁，可以通过 pymysql 模块使用 SQL 语句操作 MySQL 数据库。

在准备好了 MySQL 服务器环境以及了解了 Python 代码操作 MySQL 数据库的基本原理之后，我们可以进行相应实践。

可以打开 Python Shell，然后尝试用 Python 代码操作 MySQL 数据库。

首先，我们应当导入 pymysql 模块，如下所示。

```
>>> import pymysql
```

导入 pymysql 模块之后，我们可以通过 pymysql.connect() 连接对应数据库，连接的格式是：pymysql.connect (host="主机名",user="账号",passwd="密码",db="数据库名")，连接后可以通过 query() 来执行对应的 SQL 语句，如果不想指定连接哪一个具体的数据库，db 参数可以省略，如下所示，我们可以连接到 MySQL，然后创建一个名为 mpydb 的数据库。

```
>>> conn=pymysql.connect(host="127.0.0.1",user="root",passwd="root")
>>> conn.query("create database mpydb")
1
```

可以看到，此时我们连接后通过 query 执行了对应的 SQL 语句：create database mpydb，此时返回结果 1，所以此时成功创建了一个新的名为 mpydb 的数据库。

我们在 MySQL 终端中可以通过 show databases; 来查看当前 MySQL 中所拥有的数据库，可以看到此时已经有了新的数据库 mpydb，如图 17-8 所示。

然后，我们可以继续通过 Python 操作 mpydb，要操作 mpydb，我们可以再连接一次 MySQL，并且连接的时候通过 db 参数指定 mpydb，如下所示，我们连接到了 mpydb 数据库，并在该数据库下创建了一个名为 mytb 的表，其中有两个字段，第一个字段为 title，可以用来存储文章的标题，第二个字段为 keywd，可以用来存储当前页面的关键词。

```
>>> conn1=pymysql.connect(host="127.0.0.1",user="root",passwd="root",db="mpydb")
>>> conn1.query("CREATE TABLE mytb(title CHAR(20) NOT NULL,keywd CHAR(30))")
0
```

我们可以通过 Python 代码试着向 mytb 表中插入一些数据，如下所示：

```
>>> conn1.query("INSERT INTO mytb(title,keywd) VALUES('first title','firstkeywd')")
1
```

然后我们可以在 MySQL 终端中通过 select 语句查看一下 mytb 表中的数据，看一下是否已经通过 Python 代码成功插入数据，如图 17-9 所示：

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| abc |
| kj_neal_mall |
| mycms |
| mpydb |
+-----+
```

图 17-8 在 MySQL 终端中查看数据库

```
mysql> select * from mytb;
+-----+-----+
| title | keywd |
+-----+-----+
| first title | firstkeywd |
+-----+-----+
1 row in set (0.00 sec)
```

图 17-9 在 MySQL 终端中查询数据库

可以看到，在终端中我已经可以看到刚才通过 Python 插入的数据，那么，我们如何在 Python 中通过 select 语句查看对应表中的数据呢？

在 Python 中通过 select 语句查看对应表中的数据可以分为 3 步进行：

- 1) 通过 cursor() 创建对应的游标 (cursor);
- 2) 通过 execute() 执行对应的 select 语句;
- 3) 通过循环 (比如 for 循环) 遍历对应的内容。

比如，如果我们想通过 Python 代码遍历 mytb 中的数据，可以使用如下代码进行：

```
>>> cs=conn1.cursor()
>>> cs.execute("select * from mytb")
```

```

1
>>> for i in cs:
    print(" 当前是第 "+str(cs.rownumber)+" 行 ")
    print(" 标题是: "+i[0])
    print(" 关键词是: "+i[1])

```

然后我们可以在 Python shell 中按回车键运行这些代码。运行结果如下所示：

```

当前是第 1 行
标题是: first title
关键词是: firstkeywd

```

可以看得得到，此时已经成功得到了对应的查询结果，此时已经可以在 Python 中通过 select 语句查看对应表中的数据了。

## 17.2 爬取内容写进 MySQL

通过上一节的学习，我们已经知道如何使用 Python 操作 MySQL 数据库了，有了这些基础之后，我们就可以在 Python 爬虫中将爬取到的关注数据自动的写入 MySQL 数据库中，这样，会让我们对数据进行的操作更加方便。

假如此时我们需要创建一个爬虫项目以自动获取新浪新闻网站中多个新闻网页中的标题和关键词等信息，然后存储到数据库中，此时具体的实现过程如下所示。

首先，我们需要创建一个爬虫项目，如下所示，我们创建了一个名为 mysqlpjt 的爬虫项目。

```

D:\Python35\myweb\part17>scrapy startproject mysqlpjt
New Scrapy project 'mysqlpjt', using template directory 'd:\\python35\\lib\\
site-packages\\scrapy\\templates\\project', created in:
D:\Python35\myweb\part17\mysqlpjt
You can start your first spider with:
cd mysqlpjt
scrapy genspider example example.com

```

创建了爬虫项目之后，我们首先需要修改该项目中的 items.py 文件，如下所示：

```

# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy

class MysqlpjtItem(scrapy.Item):

```

```
# define the fields for your item here like:
```

```
# name = scrapy.Field()
```

```
# 建立 name 存储网页标题
```

```
name=scrapy.Field()
```

```
# 建立 keywd 存储网页关键词
```

```
keywd=scrapy.Field()
```

然后我们还需要修改 `pipelines.py` 文件，将对应的信息存储到数据库中主要是在该文件中编写实现的。需要注意的是，我们事先需要设计好 MySQL 数据库，然后根据设计好的数据库进行相应的处理，在此我们可以使用 17.1 节中设计的 MySQL 数据库进行操作。

我们可以将 `pipelines.py` 文件修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-
```

```
import pymysql
```

```
# Define your item pipelines here
```

```
#
```

```
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
```

```
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html
```

```
class MysqlpjtPipeline(object):
```

```
    def __init__(self):
```

```
        # 刚开始时连接对应数据库
```

```
        self.conn=pymysql.connect(host="127.0.0.1", user="root", passwd="root",
```

```
        db="mypydb")
```

```
    def process_item(self, item, spider):
```

```
        # 将获取到的 name 和 keywd 分别赋给变量 name 和变量 key
```

```
        name=item["name"][0]
```

```
        key=item["keywd"][0]
```

```
        # 构造对应的 sql 语句
```

```
        sql="insert into mytb(title,keywd) VALUES('"+name+"','"+key+"')"
```

```
        # 通过 query 实现执行对应的 sql 语句
```

```
        self.conn.query(sql)
```

```
        return item
```

```
    def close_spider(self, spider):
```

```
        self.conn.close()
```

然后我们还需要通过设置文件 `settings.py` 启用对应的 pipelines，如下所示：

```
# Configure item pipelines
```

```
# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html
```

```
ITEM_PIPELINES = {
```

```
    'mysqlpjt.pipelines.MysqlpjtPipeline': 300,
```

```
}
```

接下来，我们还需要创建和编写对应的爬虫文件来实现具体的网页的爬取。

我们先基于 `crawl` 爬虫模板创建一个名为 `weiwei` 的爬虫文件，如下所示：

```
D:\Python35\myweb\part17\mysqlpjt>scrapy genspider -t crawl weiwei sina.com.cn
```



```
Created spider 'weiwei' using template 'crawl' in module:
Mysqlepjt.spiders.weiwei
```

可以观察新浪新闻中多个新闻网页的 URL 地址, 类似为如下所示(当然读者在阅读本书的时候可能新浪新闻的网址结构发生了变化, 如果发生了变化, 读者可以自行观察其规律并总结出对应的正则表达式):

```
http://news.sina.com.cn/c/nd/2016-09-27/doc-ixfwewww1642238.shtml
http://mil.news.sina.com.cn/china/2016-09-27/doc-ixfwewmf2344634.shtml
http://finance.sina.com.cn/review/jcgc/2016-09-26/doc-ixfwewmc5580764.shtml
```

可以看得到, 我们如果要提取这些推荐的新闻网页的链接, 可以通过正则表达式 “.\*?[0-9]{4}.[0-9]{2}.[0-9]{2}.doc-.\*?shtml” 去匹配。

接下来我们可以编写创建的爬虫 `weiwei.py`, 如下所示, 关键部分已给出注释:

```
# -*- coding: utf-8 -*-
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule

from mysqlepjt.items import MysqlepjtItem

class WeiweiSpider(CrawlSpider):
    name = 'weiwei'
    allowed_domains = ['sina.com.cn']
    start_urls = ['http://news.sina.com.cn/']

    rules = (
        Rule(LinkExtractor(allow=('.*?[0-9]{4}.[0-9]{2}.[0-9]{2}.doc-.*?shtml'),
            allow_domains=('sina.com.cn')), callback='parse_item', follow=True),
    )

    def parse_item(self, response):
        i = MysqlepjtItem()
        # 通过 XPath 表达式提取网页标题
        i["name"] = response.xpath("/html/head/title/text()").extract()
        # 通过 XPath 表达式提取网页的关键词
        i["keywd"] = response.xpath("/html/head/meta[@name='keywords']/@content").extract()
        return i
```

编写好爬虫文件 `weiwei.py` 之后, 我们可以通过 `scrapy crawl weiwei` 运行该爬虫文件, 但会出现如下提示:

```
2016-09-27 11:28:43 [scrapy] DEBUG: Forbidden by robots.txt: <GET http://www.sina.com.cn/>
```

此时可以看到，我们的爬虫受到了新浪新闻服务器中 robots.txt 文件的限制。我们可以在爬虫中的设置文件 settings.py 中进行相应的修改，将 ROBOTSTXT\_OBEY 部分修改为：

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = False
```

然后再次运行该爬虫 weiwei.py，发现此时会出现如下所示的编码问题：

```
UnicodeEncodeError: 'latin-1' codec can't encode characters in position 38-39:
ordinal not in range(256)
```

这时，可以通过修改 pymysql 模块的源码来解决该编码问题。

我们可以找到 pymysql 模块的安装目录，在笔者的电脑中，pymysql 模块对应的安装目录为“D:\Python35\Lib\site-packages\pymysql”。我们打开该目录，会发现有一个名为 connections.py 的文件，如图 17-10 所示：

然后我们通过编辑器打开该文件，使用 Ctrl+F 搜索对应的关键词“charset=”，发现如图 17-11 所示的代码：

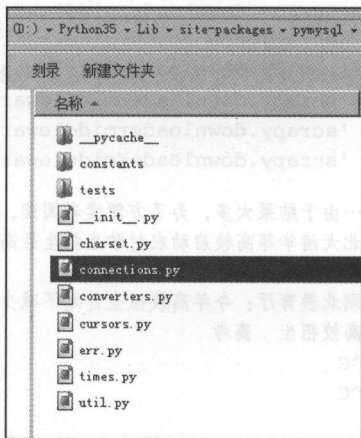


图 17-10 connections.py 文件所在位置图示

代码中，charset 参数可以设置对应的编码，只是我们可以将编码设置为“utf8”（注意此时设置的是 utf8 而不是 utf-8），所以对应的代码我们可以修改为如下所示：

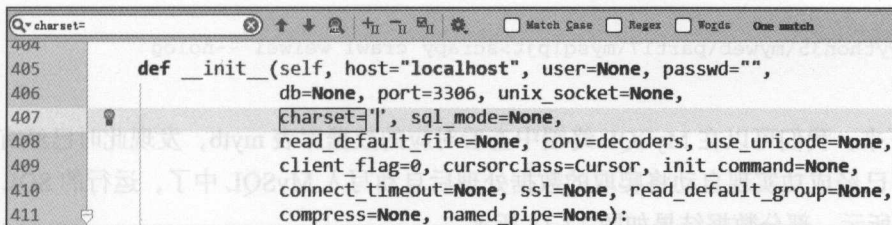


图 17-11 使用 ctrl+f 搜索到对应的代码图示

```
def __init__(self, host="localhost", user=None, passwd="",
             db=None, port=3306, unix_socket=None,
             charset='utf8', sql_mode=None,
             read_default_file=None, conv=decoders, use_unicode=None,
             client_flag=0, cursorclass=Cursor, init_command=None,
             connect_timeout=None, ssl=None, read_default_group=None,
             compress=None, named_pipe=None):
```

修改好代码之后，就设置好 pymysql 的编码了，接下来我们关闭该文件，重新运行爬虫

文件 `weiwei.py`，如下所示：

```
D:\Python35\myweb\part17\mysqlpjt>scrapy crawl weiwei
2016-09-27 13:25:52 [scrapy] INFO: Scrapy 1.1.0rc3 started (bot: mysqlpjt)
2016-09-27 13:25:52 [scrapy] INFO: Overridden settings: {'BOT_NAME': 'mysqlpjt',
'NEWSPIDER_MODULE': 'mysqlpjt.spiders', 'SPIDER_MODULES': ['mysqlpjt.spiders']}
2016-09-27 13:25:52 [scrapy] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats', 'scrapy.extensions.logstats.LogStats']
2016-09-27 13:25:52 [scrapy] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.httppauth.HttpAuthMiddleware',
'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
'scrapy.downloadermiddlewares.retry.RetryMiddleware',
'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',
'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',
'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware',
```

…由于结果太多，为了方便读者阅读，在此省略大量结果代码…

北大清华等高校启动农村学生招生最高降至一本 | 清华北大 | 高校招生 | 农村考生 \_ 新浪新闻

清华北大，高校招生，农村考生

湖北教育厅：今年高校招生计划不减少 | 高校招生 | 高考 \_ 新浪新闻

高校招生，高考

^C

^C

可以看到，此时运行爬虫已经不报错了，可以一直自动爬行下去，我们在测试的时候可以使用 `ctrl+c` 强制退出爬虫的运行，当然如果我们不想看这些运行时出现的信息，可以在爬虫运行的时候加上 `--nolog` 参数，即使用以下指令进行运行（需要注意的是，运行爬虫的时候由于要自动操作数据库，所以需要保证此时数据库为开启状态）：

```
D:\Python35\myweb\part17\mysqlpjt>scrapy crawl weiwei --nolog
^C
```

接下来，我们可以在 MySQL 终端中查看对应的数据库表 `mytb`，发现此时已经有了大量的数据，已经成功实现自动将爬取的数据处理后自动写入 MySQL 中了，运行的 SQL 语句如图 17-12 所示，部分数据结果如图 17-13 所示。

```
mysql> select * from mytb;
```

图 17-12 查询数据库中表 `mytb` 的数据

```
1  惠普发布分析前最后财报:惠普:财报:第四  1  惠普:财报:第四财季
2  知读会:“中二”钢铁侠的成长史!马斯克:  2  马斯克、伊隆·马斯克传,知读会
3  1251 rows in set (0.07 sec)
```

图 17-13 表 `mytb` 中的部分数据

此时，运行了大约十分钟，可以看到已经爬取了 1251 个网页的内容，如果我们不手动终止，其仍然可以自动地爬行下去。

## 17.3 小结

1) 在 Python2.X 中操作 MySQL 数据库可以通过 Python 的 MySQLdb 模块实现, 在 Python3.X 中我们可以使用 pymysql 模块来通过 Python 代码操作 MySQL 数据库。

2) 为了方便在本地进行测试, 我们需要在本地搭建好 MySQL 数据库服务器, 搭建的方式有很多, 比如可以使用 phpStudy 集成工具来安装 MySQL 服务器, phpStudy 集成工具可以直接在互联网中进行搜索并下载。

3) 使用 MySQL 终端工具可以通过 SQL 语句直接操作 MySQL 数据库, 而如果要使用 Python 代码操作 MySQL 数据库, 则需要通过一个中间的桥梁进行, pymysql 模块就是 Python 代码操作 MySQL 数据库的桥梁, 可以通过 pymysql 模块使用 SQL 语句操作 MySQL 数据库。

项目实战篇

• 第 18 章 博客文章爬虫项目

• 第 19 章 图片爬取项目

• 第 20 章 微博转发项目



## 第四篇 Part 4

# 项目实战篇

接下来，我们会与大家一起学习博客类爬虫项目的开发。通过本阶段的学习，我们应当掌握文章类网站爬虫项目的开发与实现，并掌握爬虫项目开发的基本流程。

## 18.1 博客类爬虫项目功能分析

爬虫项目开发的第一步，首先需要对我们想要实现的爬虫项目的功能进行定位和分析，即进行需求分析工作。

本项目中，我们需要建立一个爬虫，实现以下功能：

- 1) 爬取博客中任意一个用户的所有博文信息。
- 2) 将博文的文章名、文章 URL、文章点击数、文章评论数等信息提取出来。
- 3) 将提取出来的文章名、文章 URL、文章点击数、文章评论数等信息自动写入 MySQL 数据库中存储。

可以发现，该爬虫可以爬取任意一个用户所有博文的相关信息并存储到数据库中。实际上，我们想对某博客网站中某个用户或全站的所有博文信息进行数据分析。作为第一步，我们无法获得相应的结构化数据，此时可以通过网络爬虫将对应的信息爬取下来，并存储到 MySQL 数据库中，作为数据分析的数据源使用。比如从文章名、文章 URL、文章点击数、文章评论数中可以分析得到某个用户或博客全站中哪些博文的内容更热门，以及博文的平均点击数、平均评论数等信息，当然，如果要获得其他信息，适当调整爬虫中提取的信息即可满足需求。

该项目的重点有：

前面我们已经掌握了 Python 网络爬虫相关的基础知识，并且也通过许多小案例进行实战的分析与讲解。我们分别学习了如何通过 Urllib 模块一步步手写 Python 网络爬虫，也学习了如何使用 Scrapy 框架来编写 Python 网络爬虫。在本篇的章节中，我们会通过一些网络爬虫项目为大家讲解 Python 网络爬虫项目的开发。

本站实目页

目取虫网类客制 章 81 页  
目取虫网类书图 章 19 章  
目取虫网类登则贴 章 05 页

# 博客类爬虫项目

接下来，我们会与大家一起学习博客类爬虫项目的开发。通过本章的学习，我们应当掌握文章类网站爬虫项目的开发与实现，并掌握爬虫项目开发的基本流程。

## 18.1 博客类爬虫项目功能分析

爬虫项目开发的第一步，首先需要对我们想要实现的爬虫项目的功能进行定位和分析，即进行需求分析工作。

本项目中，我们需要建立一个爬虫，实现以下功能：

- 1) 爬取博客中任意一个用户的所有博文信息。
- 2) 将博文的文章名、文章 URL、文章点击数、文章评论数等信息提取出来。
- 3) 将提取出来的文章名、文章 URL、文章点击数、文章评论数等信息自动写入 MySQL 数据库中存储。

可以发现，该爬虫项目的主要目的是将博客中用户所有博文的相关信息提取出来并存储到数据库中。实际上，该项目会用得比较多，比如我们想对某博客网络中某个用户或全站的博文信息进行数据分析。作为第三方，我们无法获得相应的结构化数据，此时可以通过网络爬虫将对应的信息爬取下来，并存储到 MySQL 数据库中，作为数据分析的数据源使用，比如从文章名、文章 URL、文章点击数、文章评论数中可以分析得到某个用户或博客全站中哪些博文的内容更热门，以及博文的平均点击数、平均评论数等信息，当然，如果要获得其他信息。适当调整爬虫中提取的信息即可满足需求。

该项目的主要难点有：

- 1) 如何提取文章点击数、文章评论数等信息。
- 2) 如何通过循环爬取某个用户所有的博文信息。
- 3) pipelines.py 中对信息的进一步处理。
- 4) Urllib 与 scrapy 的配合使用。
- 5) XPath 表达式与正则表达式的配合使用。
- 6) 在 Scrapy 中模拟浏览器进行爬取。

18.2 博客类爬虫项目实现思路

该项目中，主要的实现思路有：

- 1) 使用 Urllib 模块手动编写相应爬虫实现。
- 2) 使用 Scrapy 框架编写相应爬虫项目实现。

如果我们要采用方式 1) 实现该项目，可以参考第 6 章，如果要采用方式 2) 实现该项目，可以参考第 12 ~ 14 章。

由于使用 Scrapy 框架实现该爬虫项目相对来说会更方便，所以在此我们总体思路是使用方式 2) 编写相应爬虫项目实现。

使用 Scrapy 框架编写相应爬虫项目的主要思路可以分为两种：

- 1) 使用基于 basic 爬虫模板的爬虫实现。
- 2) 使用基于 crawl 爬虫模板的自动爬虫实现。

在本项目中，我们将使用基于 basic 爬虫模板的爬虫实现，如果读者想使用基于 crawl 爬虫模板的自动爬虫实现，也可以参考第 16 章进行相关尝试。

18.3 博客类爬虫项目编写实战

接下来我们将为大家逐步分析并实现本爬虫项目的编写。

首先，我们需要设计好本项目中需要用到的 MySQL 数据库，以存储文章名、文章 URL、点击数、评论数等信息。

本项目中的 MySQL 数据库设计如表 18-1 所示，其中数据库名为 hexun，表名为 myhexun。

表 18-1 数据库中的表 myhexun 的设计

字段名	字段含义	字段类型及大小	是否为主键
id	文章的 id	INT ( 10 )	是
name	文章名	VARCHAR ( 30 )	否
url	文章 URL	VARCHAR ( 100 )	否
hits	文章点击数	INT ( 15 )	否
comment	文章评论数	INT ( 15 )	否



设计好数据库后，可以通过相关 SQL 语句建立设计好的数据库。

首先我们应该创建数据库 hexun，SQL 语句如下所示：

```
Create database hexun;
```

创建数据库之后，需要选择该数据库并在该数据库下创建对应的表格，SQL 语句如下所示：

```
Use hexun;
Create table myhexun(id int(10) auto_increment primary key not null,name
varchar(30),url varchar(100),hits int(15),comment int(15));
```

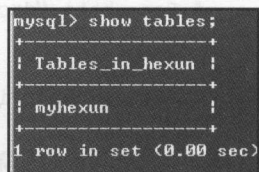
创建好表格后，我们可以通过“show tables;”语句验证一下刚刚创建的表格是否创建成功，对应的 SQL 语句及执行结果如图 18-1 所示。

可以看到，此时在数据库 hexun 下已经有了表 myhexun。

设计并建立好数据库后，我们需要基于 Scrapy 框架编写我们的爬虫项目。

首先，可以在 CMD 命令行中创建对应的爬虫项目 hexunpjt，如下所示：

```
D:\Python35\myweb\part18>scrapy startproject hexunpjt
New Scrapy project 'hexunpjt', using template directory 'd:\python35\lib\
site-packages\scrapy\templates\project', created in:
D:\Python35\myweb\part18\hexunpjt
You can start your first spider with:
cd hexunpjt
scrapy genspider example example.com
```



```
mysql> show tables;
+-----+
| Tables_in_hexun |
+-----+
| myhexun         |
+-----+
1 row in set (0.00 sec)
```

图 18-1 展示当前数据库下的所有表

创建好爬虫项目之后，我们可以通过编辑器先编写爬虫项目中的 items.py 文件，我们将 items.py 文件修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy

class HexunpjtItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    # 建立 name 存储文章名
    name = scrapy.Field()
    # 建立 url 存储文章网址
```

```

url= scrapy.Field()
# 建立 hits 存储文章阅读数
hits= scrapy.Field()
# 建立 comment 存储文章评论数
comment= scrapy.Field()

```

编写好 items.py 文件之后，还需要编写 pipelines.py 文件对爬取到的信息进一步处理，我们将 pipelines.py 文件修改为如下所示，关键部分已给出注释：

```

# -*- coding: utf-8 -*-
import pymysql
# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

class HexunpjtPipeline(object):

    def __init__(self):
        # 刚开始时连接对应数据库
        self.conn=pymysql.connect(host="127.0.0.1", user="root", passwd="root", db
            ="hexun")

    def process_item(self, item, spider):
        # 每一个博文列表页中包含多篇博文的信息，我们可以通过 for 循环一次处理各博文的信息
        for j in range(0, len(item["name"])):
            # 将获取到的 name、url、hits、comment 分别赋给各变量
            name=item["name"][j]
            url=item["url"][j]
            hits=item["hits"][j]
            comment=item["comment"][j]
            # 构造对应的 sql 语句，实现将获取到的对应数据插入数据库中
            sql="insert into myhexun(name,url,hits,comment) VALUES('"+name+"',"
                "+url+"','"+hits+"','"+comment+"')"
            # 通过 query 实现执行对应的 sql 语句
            self.conn.query(sql)
        return item

    def close_spider(self, spider):
        # 最后关闭数据库连接
        self.conn.close()

```

接下来，我们还需要编写 settings.py 文件进行相应的配置。

首先，需要开启 ITEM\_PIPELINES，将设置文件中的 ITEM\_PIPELINES 修改为如下所示：

```

# Configure item pipelines
# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html

```

```
ITEM_PIPELINES = {
    'hexunpjt.pipelines.HexunpjtPipeline': 300,
}
```

开启 ITEM\_PIPELINES 之后, 为了避免爬虫被对方服务器通过我们的 Cookie 信息识别从而将我们的爬虫禁止, 可以关闭使用 Cookie, 将设置文件中的 COOKIES\_ENABLED 信息修改为如下所示:

```
# Disable cookies (enabled by default)
COOKIES_ENABLED = False
```

随后, 为了避免对方服务器的 robots.txt 文件对我们的爬虫进行限制, 可以在文件中设置不遵循 robots 协议进行爬取, 我们将设置文件中的 ROBOTSTXT\_OBEY 部分设置为如下所示:

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = False
```

配置好设置文件 settings.py 之后, 我们可以在本项目中创建一个爬虫, 在 CMD 命令行中使用以下指令进行创建:

```
D:\Python35\myweb\part18\hexunpjt>scrapy genspider -t basic myhexunspd hexun.com
Created spider 'myhexunspd' using template 'basic' in module:
    hexunpjt.spiders.myhexunspd
```

在此, 我们基于 basic 模板创建了一个名为 myhexunspd 的爬虫, 接下来, 我们需要编写该爬虫实现网页的爬取。

在编写爬虫之前, 先来分析一下如何编写该爬虫。

首先, 打开某博客中任意一个用户的博文列表页, 比如我们打开的网址是: <http://fjrs168.blog.hexun.com/>。打开后, 出现如图 18-2 所示界面。

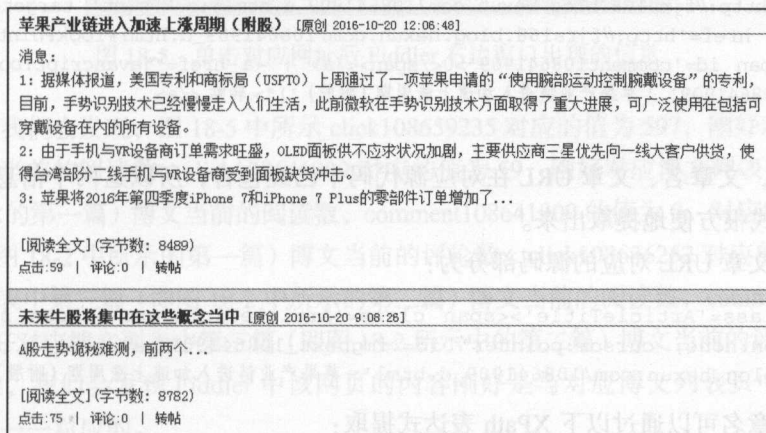
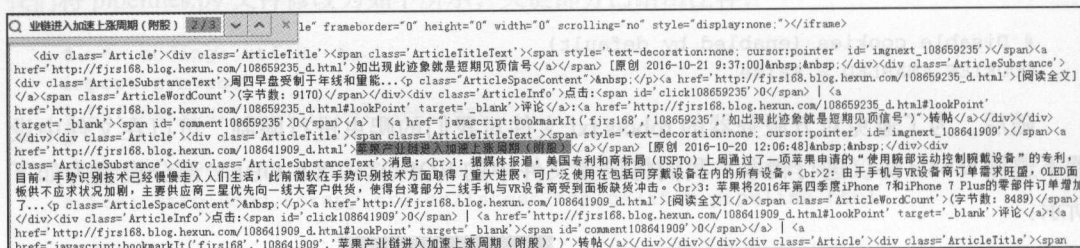


图 18-2 用户 fjrs168 的博文列表页

我们需要将这个页面中的所有博文的文章名、文章 URL、文章点击数、文章评论数等信息提取出来。可以以该页面中的任意一篇文章进行分析,通过个例的规律然后发现其中的普遍规律,比如我们以图 18-2 所示中第一篇(实际在博文列表中是第二篇)文章《苹果产业链进入加速上涨周期(附股)》为例进行分析。

查看该网页的源代码,对应的源代码如图 18-3 所示。





文章 URL 可以通过如下 XPath 表达式提取：

```
 "//span[@class='ArticleTitleText']/a/@href"
```

可以看到，源代码中没有包含文章的点击数和阅读数等信息，因为这些信息是通过 javascript 脚本动态获取的。此时我们可以使用 Fiddler 工具进行分析。

打开 Fiddler 刷新该博客网页，可以发现在加载的过程中触发的网址中出现如图 18-4 所示网址。

单击该网址，发现 Fiddler 右边出现如图 18-5 所示信息（Response 窗口中切换到 TextView 标签）：

#	Result	Protocol	Host	URL
1	200	HTTP	nsclick.baidu.com	/v.gif?pid=307&type
2	200	HTTP	19940007.blog.hexu...	/
3	200	HTTP	login.tool.hexun.com	/rest/IsUserSelf.asp
4	200	HTTP	click.tool.hexun.com	/linkclick.aspx?blogid

图 18-4 Fiddler 中触发的网址

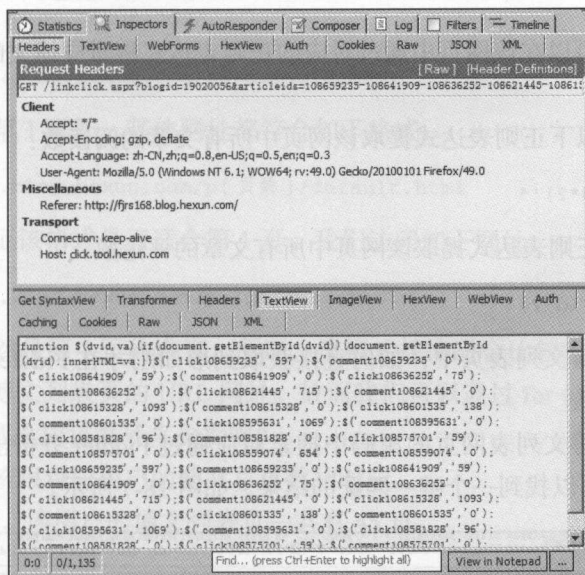


图 18-5 单击对应网址后 Fiddler 右边窗口出现的信息

通过分析我们会发现，图 18-5 中所示 click108659235 对应的值为 597，刚好对应博文列表中第一篇博文当前的阅读数，click108641909 对应的值为 59，刚好对应博文列表中第二篇（即图 18-2 中所示的第一篇）博文当前的阅读数，comment108641909 的值为 0，对应的是博文列表中第二篇（即图 18-2 中所示的第一篇）博文当前的评论数。click108636252 对应的值为 75，刚好对应博文列表中第三篇（即图 18-2 中所示的第二篇）博文当前的阅读数，comment108636252 的值为 0，刚好对应博文列表中第三篇（即图 18-2 所示中的第二篇）博文当前的评论数。

依次类推，我们会发现 Fiddler 中该网页的内容刚好是与对应博文列表页中的博文的点击数和阅读数一一对应的。

复制 Fiddler 中该网址，如下所示：

```
http://click.tool.hexun.com/linkclick.aspx?blogid=19020056&articleids = 108659235
- 108641909 - 108636252 - 108621445 - 108615328 - 108601535 - 108595631 -
108581828 - 108575701 - 108559074 - 108659235 - 108641909 - 108636252 - 108621445
- 108615328 - 108601535 - 108595631 - 108581828 - 108575701 - 108559074
```

然后通过浏览器打开该网址, 可以看到该网址的网页内容就是刚刚在 Fiddler 中查看的网页的内容, 此时网页上的内容与对应的博文列表页中的博文的点击数和阅读数一一对应。该网址的网页内容如图 18-6 所示。

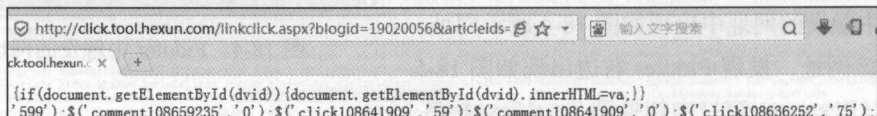


图 18-6 存储博文的点击数和阅读数的网页信息

也就是说, 我们可以在该网页中通过对应的正则表达式将对应博文的阅读数和评论数提取出来。

比如, 可以通过以下正则表达式提取该网页中所有文章的阅读数:

```
"click\d*?', '(\d*?)'"
```

还可以通过以下正则表达式提取该网页中所有文章的评论数:

```
"comment\d*?', '(\d*?)'"
```

那么, 在对应的博文列表页中, 如何直接获取到形如刚才在 Fiddler 中查看的存储评论数与阅读数的网址呢?

同样, 可以在该博文列表网页的源码中搜索对应通过 Fiddler 分析出来的存储评论数与阅读数的网址, 这时可以找到一个唯一匹配的结果, 如图 18-7 所示。

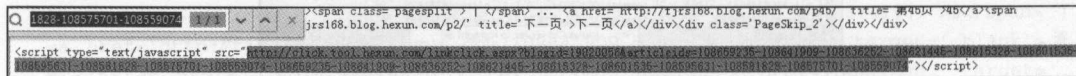


图 18-7 博文列表页的源码中搜索存储评论数与阅读数的网址出现的结果

将图中对应源代码复制出来, 如下所示:

```
<script type="text/javascript" src="http://click.tool.hexun.com/linkclick.aspx?blogid = 19020056&articleids = 108659235 - 108641909 - 108636252 -
108621445 - 108615328 - 108601535 - 108595631 - 108581828 - 108575701 -
108559074 - 108659235 - 108641909 - 108636252 - 108621445 - 108615328 -
108601535 - 108595631 - 108581828 - 108575701 - 108559074"></script>
```

然后通过如下正则表达式将对应的存储评论数与阅读数的网址信息提取出来:

```
'<script type="text/javascript" src="(http://click.tool.hexun.com/.*)">'
```

通过以上分析, 我们发现, 如果要获取当前博文列表页中对应博文的点击数和评论数,

可以这样做：

- 1) 通过正则表达式获取对应存储博文的点击数和评论数的网址。
- 2) 通过 `urllib.request` 爬取对应存储博文的点击数和评论数的网址中的数据。
- 3) 使用正则表达式将博文的点击数和评论数提取出来。

可以看到，在第 2) 步中我们在 Scrapy 框架的爬虫项目中使用了 Urllib 模块，此时，Urllib 模块与 Scrapy 框架可以互相配合使用。

至此，我们成功提取了一个博文列表页中的所有博文的文章名、文章 URL、文章点击数、文章评论数等信息。

实际上，一个用户可能有多页博文，如何爬取一个用户的所有博文信息呢？

可以观察该用户的不同页博文的 URL 网址的变化，变化如下：

```
http://fjrs168.blog.hexun.com/
http://fjrs168.blog.hexun.com/p2/default.html
http://fjrs168.blog.hexun.com/p3/default.html
```

我们发现，除第 1 页外，其他网址都符合如下格式：

```
http://fjrs168.blog.hexun.com/p[ 页数 ]/default.html
```

此时，可以验证该格式是否适合第 1 页，我们访问如下网址：

```
http://fjrs168.blog.hexun.com/p1/default.html
```

发现出现的仍然是第 1 页的内容，所以，该格式规律适合比较特殊的第 1 页，即该格式适合该用户的所有博文列表页的 URL 网址。故而我们可以通过 for 循环依次爬取对应用户的所有博文列表页。那么应该循环多少次呢？

循环的次数应该跟对应用户的博文总页数相对应。

如图 18-8 所示，该用户的博文列表页一共有 45 页。

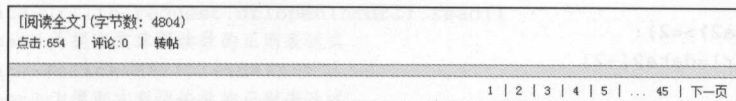


图 18-8 博文列表页中的页码展示

如何通过表达式提取对应用户的博文总页数呢？

可以查看该博文列表页中与“45”这个关键词相关的对应源代码，搜索“45”，如图 18-9 所示。

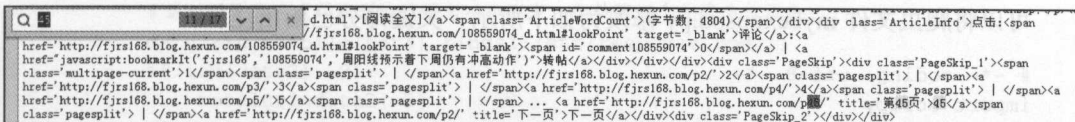


图 18-9 博文列表页源码中搜索“45”的结果

将相关源代码复制出来如下所示：

```
...
<a href='http://fjrs168.blog.hexun.com/p4/'>4</a><span class='pagesplit'> | </span><a href='http://fjrs168.blog.hexun.com/p5/'>5</a><span class='pagesplit'> | </span> ... <a href='http://fjrs168.blog.hexun.com/p45/' title=' 第 45 页 '>45</a>
...
```

我们会发现，45 在形如 “blog.hexun.com/p45/default.html” 的网址中。在源码中搜索类似的网址格式 “blog.hexun.com/p”，发现一共有 6 个匹配结果，如图 18-10 所示。

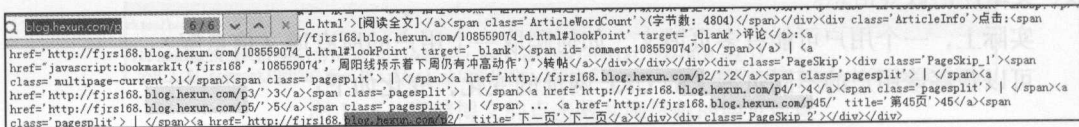


图 18-10 博文列表页源码中搜索 “blog.hexun.com/p” 的结果

而 45 刚好在倒数第二个网址中，我们发现倒数第一个网址为下一页的网址，而倒数第二个网址就是最后一页的网址，此时若一个用户的博文列表页出现换页的情况，那么形如上面的网址中倒数第二个网址都为最后一页的网址，即倒数第二个网址都会存储对应用户博文列表页的总页数。

所以，我们可以先通过如下正则表达式获取所有符合格式 “blog.hexun.com/p[数字]” 的网址：

```
"blog.hexun.com/p(.*?)/"
```

然后再通过 list[-2] 获取倒数第二个数据，得到文章列表页的总页数，就可以知道 for 循环的次数了。

当然，如果一个用户的文章量比较少，只有 1 页，那么此时显然无法得到总页数，为了兼容这种特殊情况，可以通过类型如下的语句进行判断：

```
if (len(data2)>=2):
    totalurl=data2[-2]
else:
    totalurl=1
```

语句中主要表明，如果通过正则表达式获取到的所有符合格式 “blog.hexun.com/p[数字]” 的网址数据大于或等于两个，说明此时有多页，页码数为列表中的倒数第二个元素，否则说明此时只有 1 页，那么就将页码数 totalurl 赋值为 1。

此时，我们已经能够实现获取任意一个用户的所有博文了。

可以将爬虫文件 myhexunspd.py 修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-
import scrapy
import re
import urllib.request
```



```

from hexunpjt.items import HexunpjtItem
from scrapy.http import Request

class MyhexunspdSpider(scrapy.Spider):
    name = "myhexunspd"
    allowed_domains = ["hexun.com"]
    # 设置要爬取的用户的 uid, 为后续构造爬取网址做准备
    uid = "19940007"
    # 通过 start_requests 方法编写首次的爬取行为
    def start_requests(self):
        # 首次爬取模拟成浏览器进行
        yield Request("http://" + str(self.uid) + ".blog.hexun.com/p1/default.html", headers
                      = {'User-Agent': "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
                        (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0"})

    def parse(self, response):
        item = HexunpjtItem()
        item['name'] = response.xpath("//span[@class='ArticleTitleText']/a/text()").extract()
        item["url"] = response.xpath("//span[@class='ArticleTitleText']/a/@href").extract()
        # 接下来需要使用 urllib 和 re 模块获取博文的评论数和阅读数
        # 首先提取存储评论数和点击数网址的正则表达式
        pat1 = '<script type="text/javascript" src="(http://click.tool.hexun.com/.*)">'
        # hcurl 为存储评论数和点击数的网址
        hcurl = re.compile(pat1).findall(str(response.body))[0]
        # 模拟成浏览器
        headers2 = ("User-Agent",
                    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML,
                    like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0")
        opener = urllib.request.build_opener()
        opener.addheaders = [headers2]
        # 将 opener 安装为全局
        urllib.request.install_opener(opener)
        # data 为对应博客列表页的所有博文的点击数与评论数数据
        data = urllib.request.urlopen(hcurl).read()
        # pat2 为提取文章阅读数的正则表达式
        pat2 = "click\d*",'(\d*)'"
        # pat3 为提取文章评论数的正则表达式
        pat3 = "comment\d*",'(\d*)'"
        # 提取阅读数和评论数数据并分别赋值给 item 下的 hits 和 comment
        item["hits"] = re.compile(pat2).findall(str(data))
        item["comment"] = re.compile(pat3).findall(str(data))
        yield item
        # 提取博文列表页的总页数
        pat4 = "blog.hexun.com/p(.*)/"
        # 通过正则表达式获取到的数据为一个列表, 倒数第二个元素为总页数
        data2 = re.compile(pat4).findall(str(response.body))
        if (len(data2) >= 2):
            totalurl = data2[-2]
        else:
            totalurl = 1

```

```

# 在实际运行中,下一行 print 的代码可以注释掉,在调试过程中,可以开启下一行 print 的代码
#print("一共 "+str(totalurl)+" 页 ")
# 进入 for 循环,依次爬取各博文列表页的博文数据
for i in range(2,int(totalurl)+1):
    # 构造下一次要爬取的 url,爬取一下页博文列表页中的数据
    nexturl="http://"+str(self.uid)+".blog.hexun.com/p"+str(i)+"/default.html"
    # 进行下一次爬取,下一次爬取仍然模拟成浏览器进行
    yield Request(nexturl,callback=self.parse,headers = {'User-Agent': 'Mozilla
/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0'})

```

如果要爬取其他用户的信息,只需要更改爬虫文件中的 uid 即可。

比如,我们需要对时寒冰的博客进行爬取,首先查看该用户的博客网址如下:

```
http://shihanbingblog.blog.hexun.com/
```

可以看到,该用户的专属域名为“shihanbingblog”,对应爬虫文件中的 uid,所以可以将爬虫文件 myhexunspd.py 中的:

```
uid = "19940007"
```

修改为:

```
uid = "shihanbingblog"
```

即可以实现爬取另一个用户的博文信息。

## 18.4 调试与运行

编写好对应的爬虫项目之后,我们可以运行该爬虫项目下的爬虫文件 myhexunspd.py,如下所示:

```
D:\Python35\myweb\part18\hexunpjt>scrapy crawl myhexunspd --nolog
```

运行完后,可以在 MySQL 数据库终端中通过 select 语句查询此时数据库中的对应博文信息,SQL 语句如图 18-11 所示,运行结果如图 18-12 所示。

```
mysql> select * from myhexun;
```

图 18-11 通过 select 语句查询此时数据库中的对应博文信息

```

9940007.blog.hexun.com/79305806_d.html      | 1 |
! 5695 ! 2010: 参政议政,我乐此不疲
9940007.blog.hexun.com/79305805_d.html      | 2 |
! 5696 ! "两会"前夕,调控政策将加码
9940007.blog.hexun.com/79305804_d.html      | 1 |
+-----+-----+
5696 rows in set (0.02 sec)

```

图 18-12 图 18-11 中指令的执行结果

可以看到,此时一共有 5696 条数据,只是我们分别爬取了用户 "19940007" 和用户 "shihan

bingblog" 的信息。

在此,我们还可以通过 select 语句查询出评论数最多的前 10 条博文,如图 18-13 所示。

除此之外,有了这些数据后,我们还可以进行很多数据分析,当然数据分析的知识体系不是本书的内容,这里不再过多讲解。

那么,如何爬取整个博客站点中所有用户的所有博文信息呢?

我们可以观察不同用户的主页的 URL 网址变化:

```
http://19940007.blog.hexun.com/  
http://26325289.blog.hexun.com/  
http://21282349.blog.hexun.com/  
...
```

可以发现,其网址的规律为 `http://[数字].blog.hexun.com/`,那么,我们可以猜测,对应的数字为博客的 ID,我们可以分别带入几个数字验证 URL 网址,发现以下网址均为不同用户的博客:

```
http://3.blog.hexun.com/  
http://199.blog.hexun.com/  
http://21282329.blog.hexun.com/  
...
```

所以,网址中前面的数字应该为用户的博客序号,可以通过 for 循环将所有博客用户的博客主页遍历完。

此时我们可以在外层再加一层 for 循环即可实现自动遍历不同的用户的博文信息,该部分的实现不难,感兴趣的读者可以自行探索,在此不进行过多讲解。

当然,若要遍历非常多次,则需要一个非常大的服务器,并且若服务器资源允许,最好可以采用分布式爬取的方式。

## 18.5 小结

1) 本项目采用 Scrapy 框架结合 Urllib 模块编写的方式实现博客类爬虫,大家需要掌握这两者结合的方式来编写爬虫项目。

2) 本项目中分别采用了正则表达式和 XPath 表达式提取信息,在实际中具体使用哪种表达式主要是看哪种表达式提取信息更为方便,这里我们主要学会在 Scrapy 爬虫项目中如何使用正则表达式和 XPath 表达式共同来提取信息。

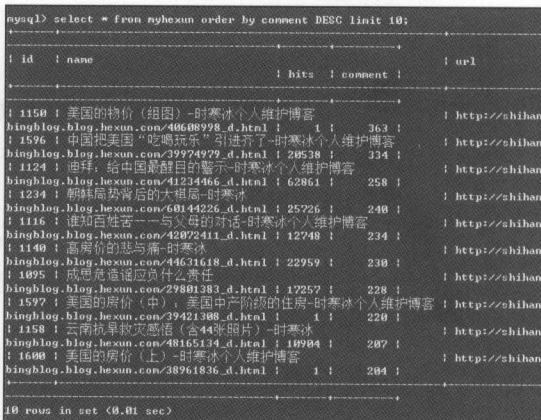


图 18-13 通过 select 语句查询出评论数最多的前 10 条博文

## 图片类爬虫项目

我们在第 6 章中已经学习过图片爬虫，但在前面学习的图片爬虫是通过 Urllib 模块完全手写实现的，在本章中，我们会以图片类爬虫项目为例，为大家讲解如何通过 Scrapy 框架实现图片爬虫项目。

### 19.1 图片类爬虫项目功能分析

有时，我们需要对互联网中的一些图片进行分析或参考，可以将这些图片爬取到本地存储起来，这样使用起来会更加方便。

假如我们现在需要做一个商品的图片设计，需要参考互联网中的一些素材，此时通过互联网一个一个网页地打开查看会比较麻烦，此时我们可以将对应网站中相关栏目下的素材图片全部爬取到本地中使用。

这里我们参考千图网中的“淘宝设计”栏目中的素材进行相关设计。

在本项目中，主要需要实现的功能有：

- 1) 获取千图网中“淘宝设计”栏目下的所有图片素材。
- 2) 将原图片素材（非缩略图）保存到本地的对应目录中。

由于文章列表页中显示的图片 URL 为对应图片的缩略图 URL 地址，而缩略图的效果不如原图的效果好，比如不够清晰等，所以我们在本项目中会爬取对应图片的原图片 URL，而缩略图的 URL 和原图的 URL 之间是有一定规律的，这个规律我们在项目的实现过程中会为大家具体分析。



## 19.2 图片类爬虫项目实现思路

为了提高项目开发的效率,避免在项目开发的过程中思路混乱,我们需要在项目开发前首先理清该项目的实现思路及实现步骤。

本项目的实现思路及实现步骤如下所示:

- 1) 对要爬取的网页进行分析,发现要获取的内容的规律,总结出提取对应数据的表达式,并且发现不同图片列表页 URL 之间的规律,总结出自动爬取各页的方式;
- 2) 创建 Scrapy 爬虫项目;
- 3) 编写好项目对应的 items.py、pipelines.py、settings.py 文件;
- 4) 创建并编写项目中的爬虫文件,实现爬取当前列表页面的所有原图片(非缩略图),以及自动爬取各图片列表页。

## 19.3 图片类爬虫项目编写实战

首先,对要爬取的网页进行分析。

打开要爬取的第一个网页,如下所示:

`http://www.58pic.com/tb`

打开后结果如图 19-1 所示。

我们需要分析如何提取该页中所有素材图片的 URL 地址。

此时我们以该页面中其中一个图片为例来总结相关规律,比如我们以“淘宝精华橄榄油护肤品促销海报 W”图片为例进行总结。查看相关源代码,如图 19-2 所示。

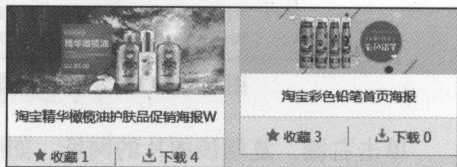


图 19-1 “淘宝设计”栏目第 1 页

我们将对应的源代码复制出来,如下所示:

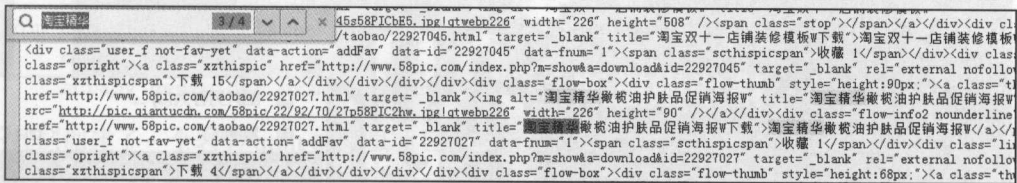


图 19-2 网页中“淘宝精华橄榄油护肤品促销海报 W”图片对应的源代码

```
...
<a class="thumb-box" href="http://www.58pic.com/taobao/22927027.html" target="_blank"></a></div><div class="flow-info2 nounderline"><p><a href="http://www.58pic.com/taobao/22927027.html" target="_blank" title=" 淘宝精华橄榄油护肤品促销海报 W 下载 "> 淘宝精华橄榄油护肤品促销海报 W</a> ...
```

可以发现，对应图片网页的网址为：

<http://www.58pic.com/taobao/22927027.html>

对应图片的缩略图网址为：

<http://pic.qiantucdn.com/58pic/22/92/70/27p58PIC2hw.jpg!qtwebp226>

为了发现原图片和缩略图网址之间的规律，我们可以打开对应图片网页的网址，然后查看对应原图片的网址。

打开图片网页的网址：<http://www.58pic.com/taobao/22927027.html>。

打开后结果如图 19-3 所示。



图 19-3 “淘宝精华橄榄油护肤品促销海报 W” 图片专属网页

单击右键，复制该图片的网址，如图 19-4 所示。



图 19-4 右键复制该图片的网址

我们得到的网址即为原图片的网址，此时得到的原图片网址如下所示：

```
http://pic.qiantucdn.com/58pic/22/92/70/27p58PIC2hw_1024.jpg
```

对比该图片的缩略图和真实图片的 URL 网址：

```
http://pic.qiantucdn.com/58pic/22/92/70/27p58PIC2hw.jpg!qtwebp226
```

```
http://pic.qiantucdn.com/58pic/22/92/70/27p58PIC2hw_1024.jpg
```

我们会发现其具有相关联系，原图片 URL 网址为缩略图 URL 网址中“.jpg”及其后面部分去掉，再加上“\_1024.jpg”。

对比其他图片的缩略图和原图片的 URL 网址，如下所示：

```
http://pic.qiantucdn.com/58pic/22/92/70/45s58PICbE5.jpg!qtwebp226
```

```
http://pic.qiantucdn.com/58pic/22/92/70/45s58PICbE5_1024.jpg
```

```
http://pic.qiantucdn.com/58pic/22/92/68/70358PICVIY.jpg!qtwebp226
```

```
http://pic.qiantucdn.com/58pic/22/92/68/70358PICVIY_1024.jpg
```

发现均具有该规律，也就是说，只要我们得到图片的缩略图的 URL 地址，就可以构造出原图片的 URL 地址。

图片缩略图对应的源代码为：

```
...

...
```

所以图片缩略图地址可以通过正则表达式找出：

```
"(http://pic.qiantucdn.com/58pic/.*)\.jpg"
```

然后就可以根据提取出来的网址构造出真实图片的网址了。

接下来，我们需要创建爬虫项目，如下所示：

```
D:\Python35\myweb\part19>scrapy startproject qtpjt
New Scrapy project 'qtpjt', using template directory 'd:\python35\lib\site-packages\
scrapy\templates\project', created in:
  D:\Python35\myweb\part19\qtpjt
You can start your first spider with:
  cd qtpjt
  scrapy genspider example example.com
```

我们创建了一个名为 qtpjt 的爬虫项目，然后可以在该爬虫项目中编写 items.py 文件，将 items.py 文件修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-
```

```
# Define here the models for your scraped items
```

```
#
```

```
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html
```

```
import scrapy
```

```
class QtpjtItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    # 建立 picurl 存储图片的网址
    picurl=scrapy.Field()
    # 建立 picid 存储图片网址中的图片名, 以方便构造本地文件名
    picid=scrapy.Field()
```

编写好 items.py 文件之后, 我们需要编写 pipelines.py 文件, 将 pipelines.py 文件修改为如下所示, 关键部分已给出注释:

```
# -*- coding: utf-8 -*-
import urllib.request
# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

class QtpjtPipeline(object):
    def process_item(self, item, spider):
        # 一个图片列表页中有多张图片, 通过 for 循环依次将图片存储到本地
        for i in range(0, len(item["picurl"])):
            thispic=item["picurl"][i]
            # 根据上面总结的规律构造出原图片的 URL 地址
            trueurl=thispic+"_1024.jpg"
            # 构造出图片在本地存储的地址
            localpath = "D:/Python35/myweb/part19/pic/" + item["picid"][i] + ".jpg"
            # 通过 urllib.request.urlretrieve() 将原图片下载到本地
            urllib.request.urlretrieve(trueurl, filename=localpath)
        return item
```

然后, 修改配置文件 settings.py, 我们将配置文件中关于 pipelines 的配置部分修改为如下所示, 这样就可以开启使用 pipelines.py 文件了:

```
# Configure item pipelines
# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    'qtpjt.pipelines.QtpjtPipeline': 300,
}
```

接下来, 我们还需要在该爬虫项目中创建对应爬虫, 如下所示:

```
D:\Python35\myweb\part19\qtpjt>scrapy genspider -t basic qtspd 58pic.com --nolog
```



修改为如下所示，关键部

```
response.body)))
?/.*/(.*?).jpg"
(response.body))
".html"
```

爬虫:

g

对应栏目下的图片了，一  
们可以在修改该爬虫项目

入口网址和 for 循环中的下一页网址 nexturl 后，将其轻松运用到该站其他栏目的图片爬取。



图 19-5 爬虫运行后爬取到的图片

## 19.5 小结

1) 由于文章列表页中显示的图片 URL 为对应图片的缩略图 URL 地址，而缩略图的效果不如原图的效果好，比如不够清晰等，所以我们在本项目中会爬取对应图片的原图片 URL，而缩略图的 URL 和原图的 URL 之间是有一定规律的，这个规律的分析方式我们应当掌握，以便在遇到类似情况的时候能够灵活运用。

2) 掌握该项目中 `urllib.request.urlretrieve()` 的使用方法。

## 模拟登录爬虫项目

在互联网中，不需要提交表单，使用静态的链接就能够访问的静态页面属于表层网页，而那些隐藏在表单后面，需要提交一定的表单或发送一定的关键词才能获取的页面称为深层网页。在互联网中深层网页比表层网页要多得多，比如，那些需要登录后才能获取到的网页就属于深层网页。

关于如何使用 GET 请求或 POST 请求获取深层网页的方式，我们在第 5 章中已经为大家提到，在本章中，我们会以模拟登录豆瓣爬虫项目为例，为大家讲解如何在 Scrapy 爬虫项目中获取深层网页。

### 20.1 模拟登录爬虫项目功能分析

在本项目中，主要实现以下功能：

- 1) 在 Scrapy 中使用 FormRequest 传递 POST 请求实现网站的登录；
- 2) 在程序中对登录过程中可能会出现验证码进行处理；
- 3) 登录后保存相关 Cookie 信息，让爬行该站其他网页的时候可以通过 Cookie 保留登录状态；
- 4) 登录成功后，爬取网站个人中心的相关信息，获取登录的用户个人中心中显示的所发表的日记信息，并且将相关信息输出出来。

本项目中的难点部分在于登录的实现、Cookie 的处理以及验证码的处理。

### 20.2 模拟登录爬虫项目实现思路

要实现该爬虫项目，首先我们需要分析出网站登录时真实的表单 POST 提交地址，然

后分析需要提交的信息及含义。随后建立好验证码处理机制，让爬虫在遇到验证码时不会崩溃，在处理了验证码之后可以继续通过 FormRequest 登录并爬取对应的网页。

实现了网页的登录之后，还需要保存对应的 Cookie 信息，让登录状态可以保持。随后，我们可以爬取登录用户的个人中心页，实现对应信息的爬取并输出。

### 20.3 模拟登录爬虫项目编写实战

首先，我们可以对登录页面进行分析。

在豆瓣网站的主页上单击“登录”，此时出现的网址如下所示：

https://www.douban.com/accounts/login?redir=https%3A//www.douban.com/mine/orders/

该网址对应的网页内容如图 20-1 所示。

如果是通过浏览器登录的话，在此输入账号、密码和验证码之后，单击“登录”按钮即可实现登录。

但如果要使用爬虫进行登录，我们则需要分析出这些表单里面账号、密码和验证码等登录信息的 POST 请求提交地址。

可以在浏览器中按“F12”，调出调试界面。

然后可以在上面的网页中输入账号、密码和验证码，并单击“登录”按钮，此时触发登录动作，让我们可以看到调试界面中出现了相应的新的请求。

如图 20-2 所示，可以看到，其中有一个请求是 POST 请求，点开该请求，可以看到右边窗口中出现了对应的网址，此时网址为 https://accounts.douban.com/login，该网址就是 POST 请求真实的提交页。

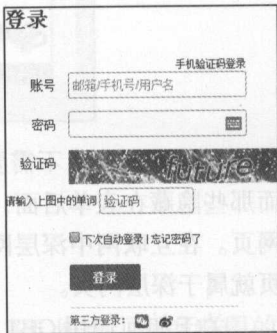


图 20-1 登录页

5 个请求, 3.25 KB, 0.90s									
前	所有	HTML	CSS	JS	XHR	字体	图像	媒体	Flash
其他	状态	方法	文件	域名	事由	消息头	Cookie	参数	
●	200	OPTIONS	collect	zhihu-web-analytics.zhihu.com	xhr	请求网址: http://www.zhihu.com/login/email			
●	200	OPTIONS	collect	zhihu-web-analytics.zhihu.com	xhr	请求方法: POST			
	200	POST	email	www.zhihu.com	js xhr	远程地址: 127.0.0.1:8888			
●	200	GET	captcha.gif?r=1475477845891&type=login	www.zhihu.com	img	状态码: 200 OK			
●	200	POST	batch	zhihu-web-analytics.zhihu.com	xhr	版本: HTTP/1.1			

图 20-2 通过 F12 分析 POST 请求真实的提交页

在浏览器中输入分析出来的网址 https://accounts.douban.com/login 并回车，出现如图 20-3 所示界面。

可以看到，该网页就是爬虫在进行登录的时候需要着重分析的网页。

查看该页面表单部分对应的源码，如下所示：



登录

手机验证码登录

账号 邮箱/手机号/用户名

密码

验证码

请输入上图中的单词 验证码

☐ 下次自动登录 | 忘记密码了

登录

图 20-3 POST 请求真实的提交网页内容

.....

```
<form id="lzform" name="lzform" method="post" onsubmit="return validateForm(this);"
action="https://accounts.douban.com/login">
<div style="display:none;">

</div>
<input name="source" type="hidden" value="None"/>
<input name="redir" type="hidden" value="https://www.douban.com/mine/orders/" />
<div class="item-right">
<a href="?redir=https://www.douban.com/mine/orders/&amp;source=None&amp;login_
type=sms">手机验证码登录 </a>
</div>
<div class="item">
<label>账号 </label>
<input id="email" name="form_email" type="text" class="basic-input"
maxlength="60" value=" 邮箱 / 手机号 / 用户名 " tabindex="1"/>
</div>
<div class="item">
<label>密码 </label>
<input id="password" name="form_password" type="password" class="basic-input"
maxlength="20" tabindex="2"/>
</div>
<!-- poTJObgLdYs | 49.221.183.15 -->

<div class="item item-captcha">
<label>验证码 </label>
<div>

<div class="captcha_block">
<span id="captcha_block" class="pl">请输入上图中的单词 </span>
<input type="text" id="captcha_field" name="captcha-solution" tabindex=3
placeholder="验证码" />
<input type="hidden" name="captcha-id" value="u46aiQZFurrVJU2Z9lQB0vS6:en"/>
</div>
```



```
</div>
</div>
<div class="item">
<label>&nbsp;</label>
<p class="remember">
<input type="checkbox" id="remember" name="remember" tabindex="4"/>
<label for="remember" class="remember"> 下次自动登录 </label>
| <a href="https://accounts.douban.com/resetpassword"> 忘记密码了 </a>
</p>
</div>
<div class="item">
<label>&nbsp;</label>
<input type="submit" value=" 登录 " name="login" class="btn-submit" tabindex="5"/>
</div>
.....
```

在上面的 HTML 代码中，表单中关于登录账号的部分相关 HTML 源码为：

```
<input id="email" name="form_email" type="text" class="basic-input"
maxlength="60" value=" 邮箱 / 手机号 / 用户名 " tabindex="1"/>
```

所以我们可以得到，登录账号对应的字段名为（input 中 name 对应的信息）：

form\_email

同样，也可以看到，表单中关于密码的部分相关 HTML 源码为：

```
<input id="password" name="form_password" type="password" class="basic-input"
maxlength="20" tabindex="2"/>
```

其中，密码对应的字段名为（input 中 name 对应的信息）：

form\_password

上面的 HTML 源代码中，表单中关于验证码的部分相关 HTML 源码为：

```

<div class="captcha_block">
<span id="captcha_block" class="pl"> 请输入上图中的单词 </span>
<input type="text" id="captcha_field" name="captcha-solution" tabindex=3
placeholder=" 验证码 " />
```

所以同样的道理可以得到，验证码对应的字段名为：

captcha-solution

遇到验证码时，需要进行相关的处理。在实际项目中，验证码的处理方式有三种：

- 1) 手动输入验证码。
- 2) 通过编写程序自动识别验证码。
- 3) 通过一些打码接口实现，让别人通过接口帮我们输入验证码，但是需要支付一定的

费用。

对于一些中小型项目，我们通过方式 1) 就能处理验证码；对于大型项目，有研发能力的团队可以通过方式 2) 处理，方式 2) 主要运用计算机图像识别方面的知识，本书主题为爬虫故不会涉及，故在此不进行过多讲解，感兴趣的读者可以自行研究；没有研发能力的团队或者团队不想花过多精力去研究计算机图像识别，可以通过第 3) 种方式解决，我们可以通过一些打码接口，比如打码兔等平台，支付一定的费用，让别人通过接口帮我们打码，从而解决验证码问题。

在此，我们将使用第 1) 种方式解决验证码问题，即登录时若没有出现要输验证码的情况，则直接自动登录，若出现要输验证码的情况，则手动输入验证码后自动登录。

所以，我们需要分析服务器中验证码图片是从哪里获取的。

可以看到，在验证码相关的 HTML 源码中，有一段是：

```

```

上面代码中的：

```
https://www.douban.com/misc/captcha?id=u46aiQZFurrVJU2Z9lQB0vS6:en&size=s
```

该网址即为服务器中验证码图片的获取网址。

可以在浏览器中输入该网址进行验证：

```
https://www.douban.com/misc/captcha?id=CjR4oegyafCKQ0VVUv9LTjpX:en&size=s
```

结果如图 20-4 所示。



图 20-4 浏览器中输入验证码网址的执行结果

可以看到，该网页中的上述网址即为验证码获取网址，所以，我们如果要得到服务器中验证码图片的网址，可以通过以下 XPath 表达式得到：

```
'//img[@id="captcha_image"]/@src'
```

然后，我们可以将验证码图片保存到本地显示，以供我们输入验证码。如果想通过接口实现验证码识别，只需将验证码图片显示给别人，具体的设置在各打码平台的文档中基本都会提到。

经过上面的分析，我们就可以通过 FormRequest 实现登录了。

登录之后，我们还需要从个人中心页中爬取相关的信息。



图 20-5 登录后的个人中心页

假如我们需要获取网页标题、个人中心页中展示的所有日记标题、日记发表时间、日记内容、日记链接等信息。

可以右键查看该页对应的 HTML 源代码，日记信息部分的源代码如下所示：

[illegible]



```

</div><div class="clear"></div>

<div class="note" id="note_584776354_short"> 这是测试 2 的正文内容。</div>

</div>
<div class="clear"></div>
<div class="mbtr2">
<div class="note-header pl2">
<a title="test7799" href="https://www.douban.com/note/584726594/" class="l1">test7799</a>
<br/>
<div><span class="pl">2016-10-03 11:46:57</span></div>
</div><div class="clear"></div>

<div class="note" id="note_584726594_short">mytest this is my test 779988</div>

</div>
<div class="clear"></div>
<div class="clear"></div>

</div>
.....

```

所以，我们可以通过如下 XPath 表达式获取日记标题：

```
xnotetitle="//div[@class='note-header pl2']/a/@title"
```

同样，可以通过如下 XPath 表达式获取日记发表时间：

```
xnotetime="//div[@class='note-header pl2']/span[@class='pl']/text()"
```

通过如下 XPath 表达式获取日记内容：

```
xnotecontent="//div[@class='mbtr2']/div[@class='note']/text()"
```

通过如下 XPath 表达式获取日记链接：

```
xnoteurl="//div[@class='note-header pl2']/a/@href"
```

有了上面的分析之后，我们可以进入爬虫项目的创建和编写了。

由于本项目的需求是只需要直接将信息输出即可，并不涉及提取信息的持久化存储等后续操作，所以我们可以不使用 items.py 文件、pipelines.py 文件，同时设置文件 settings.py 保持默认设置即可。所以在下面的项目实现中，我们可以不编写修改 items.py、pipelines.py、settings.py 等文件。

首先，创建一个名为 loginpjt 的爬虫项目，如下所示：

```
D:\Python35\myweb\part20>scrapy startproject loginpjt
New Scrapy project 'loginpjt', using template directory 'd:\\python35\\lib\\
site-packages\\scrapy\\templates\\project', created in:
D:\Python35\myweb\part20\loginpjt
You can start your first spider with:
cd loginpjt
scrapy genspider example example.com
```

创建好爬虫项目之后，可以在 CMD 命令行中进入爬虫项目对应的文件夹，然后在该爬虫项目下基于 basic 爬虫模板创建一个名为 loginspd 的爬虫，如下所示：

```
D:\Python35\myweb\part20\loginpjt>scrapy genspider -t basic loginspd douban.com
Created spider 'loginspd' using template 'basic' in module:
loginpjt.spiders.loginspd
```

然后，可以通过编辑器编写该爬虫项目下的爬虫文件 loginspd.py，我们将该爬虫文件修改为如下所示，关键部分已给出注释：

```
# -*- coding: utf-8 -*-
import scrapy
import urllib.request
from scrapy.http import Request, FormRequest

class LoginspdSpider(scrapy.Spider):
    name = "loginspd"
    allowed_domains = ["douban.com"]
    # 设置头信息变量，供下面的代码中模拟成浏览器爬取
    header = {"User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36 SE 2.X MetaSr 1.0"}
    # 编写 start_requests() 方法，第一次会默认调取该方法中的请求
    def start_requests(self):
        # 首先爬一次登录页，然后进入回调函数 parse()
        return [Request("https://accounts.douban.com/login", meta={"cookiejar": 1},
            callback=self.parse)]

    def parse(self, response):
        # 获取验证码图片所在地址，获取后赋给 captcha 变量，此时 captcha 为一个列表
        captcha=response.xpath('//img[@id="captcha_image"]/@src').extract()
        # 因为登录时有时网页有验证码，有时网页没有验证码
        # 所以需要判断此时是否需要输入验证码，若 captcha 列表中有元素，说明有验证码信息
        if len(captcha)>0:
            print(" 此时有验证码 ")
            # 设置将验证码图片存储到本地的本地地址
            localpath="D:/Python35/myweb/part20/loginpjt/captcha.png"
            # 将服务器中的验证码图片存储到本地，供我们在本地直接进行查看
            urllib.request.urlretrieve(captcha[0], filename=localpath)
            print(" 请查看本地图片 captcha.png 并输入对应验证码： ")
            # 通过 input() 等待我们输入对应的验证码并赋给 captcha_value 变量
            captcha_value=input()
```

```

# 设置要传递的 post 信息
data={
    # 设置登录账号, 格式为账号字段名: 具体账号
    "form_email": "weisuen007@163.com",
    # 设置登录密码, 格式为密码字段名: 具体密码, 读者需要将账号密码换成自己的
    # 因为笔者完成该项目后已经修改密码
    "form_password": "weijc7789",
    # 设置验证码, 格式为验证码字段名: 具体验证码
    "captcha-solution": captcha_value,
    # 设置需要转向的网址, 由于我们需要爬取个人中心页, 所以转向个人中心页
    "redir": "https://www.douban.com/people/151968962/",
}

# 否则说明 captcha 列表中没有元素, 即此时不需要输入验证码信息
else:
    print(" 此时没有验证码 ")
    # 设置要传递的 post 信息, 此时没有验证码字段
    data={
        "form_email": "weisuen007@163.com",
        "form_password": "weijc7789",
        "redir": "https://www.douban.com/people/151968962/",
    }

print(" 登录中...")
# 通过 FormRequest.from_response() 进行登录
return [FormRequest.from_response(response,
    # 设置 cookie 信息 meta={"cookiejar": response.meta["cookiejar"]},
    # 设置 headers 信息模拟成浏览器
    headers=self.header,
    # 设置 post 表单中的数据
    formdata=data,
    # 设置回调函数, 此时回调函数为 next()
    callback=self.next,
)]

def next(self, response):
    print(" 此时已经登录完成并爬取了个人中心的数据 ")
    # 此时 response 为个人中心网页中的数据
    # 以下通过 XPath 表达式分别提取个人中心中该用户的相关信息
    # 网页标题 XPath 表达式
    xtitle="/html/head/title/text()"
    # 日记标题 XPath 表达式
    xnotetitle="//div[@class='note-header pl2']/a/@title"
    # 日记发表时间 XPath 表达式
    xnotetime="//div[@class='note-header pl2']//span[@class='pl']/text()"
    # 日记内容 XPath 表达式
    xnotecontent="//div[@class='mbtr2']/div[@class='note']/text()"
    # 日记链接 XPath 表达式
    xnoteurl="//div[@class='note-header pl2']/a/@href"

    # 分别提取网页标题、日记标题、日记发表时间、日记内容、日记链接
    title=response.xpath(xtitle).extract()
    notetitle = response.xpath(xnotetitle).extract()
    notetime = response.xpath(xnotetime).extract()

```

```

notecontent = response.xpath(xnotecontent).extract()
noteurl = response.xpath(xnoteurl).extract()
print(" 网页标题是: "+title[0])
# 可能有多篇日记, 通过 for 循环依次遍历
for i in range(0,len(notetitle)):
    print(" 第 "+str(i+1)+" 篇文章的信息如下:")
    print(" 文章标题为: "+notetitle[i])
    print(" 文章发表时间为: " + notetime[i])
    print(" 文章内容为: " + notecontent[i])
    print(" 文章链接为: " + noteurl[i])
    print("-----")

```

编写好该爬虫之后, 我们可以在该爬虫项目中运行该爬虫实现相关的功能。

## 20.4 调试与运行

我们可以在 CMD 命令行中进入该爬虫项目对应的目录, 通过如下指令运行上面编写的爬虫 loginspd:

```
D:\Python35\myweb\part20\loginpjt>scrapy crawl loginspd --nolog
```

执行对应的指令后, 会出现如下提示信息:

```
D:\Python35\myweb\part20\loginpjt>scrapy crawl loginspd --nolog
```

此时有验证码

请查看本地图片 captcha.png 并输入对应验证码:

可以看到, 此时登录网页中需要输入对应的验证码, 所以在这里会等待我们输入相关的验证码。

查看本地文件夹 (该文件夹在爬虫文件的 localpath 中进行设置) D:\Python35\myweb\part20\loginpjt, 发现多出了如图 20-6 所示的一个名为 captcha.png 的图片文件。

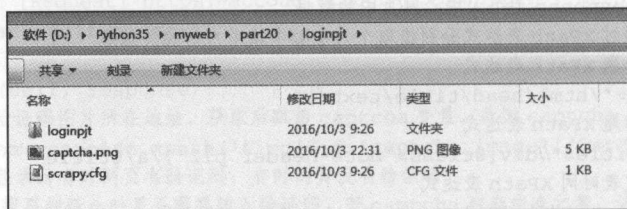


图 20-6 对应本地文件夹下出现的验证码图片

该图片文件即为验证码图片, 查看该图片的内容, 发现如图 20-7 所示的验证码信息。

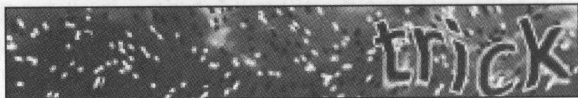


图 20-7 本地中验证码图片对应的信息



在CMD命令行中等待输入信息的地方输入“trick”，然后回车，即可出现如下执行结果：

```
D:\Python35\myweb\part20\loginpjt>scrapy crawl loginspd --nolog
```

此时有验证码

请查看本地图片 captcha.png 并输入对应验证码：

trick

登录中...

此时已经登录完成并爬取了个人中心的数据

网页标题是：

韦老师

第1篇文章的信息如下：

文章标题为：测试2

文章发表时间为：2016-10-03 19:31:40

文章内容为：这是测试2的正文内容。

文章链接为：<https://www.douban.com/note/584776354/>

第2篇文章的信息如下：

文章标题为：test7799

文章发表时间为：2016-10-03 11:46:57

文章内容为：mytest this is my test 779988

文章链接为：<https://www.douban.com/note/584726594/>

可以看到，此时已经成功实现了登录并且爬取了登录后个人中心页中的相关信息，如果登录不成功或者Cookie处理不成功，则不会爬取到登录后的对应的个人日记信息。

每次登录的时候如果需要对应的验证码，爬虫会进入等待输入的状态，我们只需要查看D:\Python35\myweb\part20\loginpjt目录下的captcha.png图片，并在cmd窗口中输入对应的验证码即可继续登录执行。当然如果觉得验证码存储到这个文件夹中不方便查看，也可以在爬虫文件中设置存储到其他的文件夹下，同时，如果资金允许或者有相应的研发团队，也可以通过验证码平台接口或者自己编写的验证码图像识别程序解决验证码的问题。

若登录时网页不需要我们输入验证码，程序也可以自动识别此时无需验证码，然后直接登录，效果如下所示：

```
D:\Python35\myweb\part20\loginpjt>scrapy crawl loginspd --nolog
```

此时没有验证码

登录中...

此时已经登录完成并爬取了个人中心的数据

网页标题是：

韦老师

第1篇文章的信息如下：

文章标题为：测试2

文章发表时间为：2016-10-03 19:31:40

文章内容为：这是测试2的正文内容。

文章链接为：<https://www.douban.com/note/584776354/>

第2篇文章的信息如下:

文章标题为: test7799

文章发表时间为: 2016-10-03 11:46:57

文章内容为: mytest this is my test 779988

文章链接为: <https://www.douban.com/note/584726594/>

## 20.5 小结

1) 通过该项目的学习, 我们应当学会 Scrapy 中 FormRequest 的使用方法, 并且能够通过 Scrapy 爬虫实现网页的自动登录。

2) 通过该项目的学习, 我们还应当学会爬虫登录时验证码的处理。因为登录网页有时需要验证码有时不需要验证码, 为了兼顾这两种情况, 我们还需要学会通过 if 语句判断网页登录时是否需要验证码。

3) 通过该项目的学习, 我们应当掌握 Scrapy 爬虫项目中 Cookie 的相关处理方式, 从而让爬虫运行时保持登录状态。

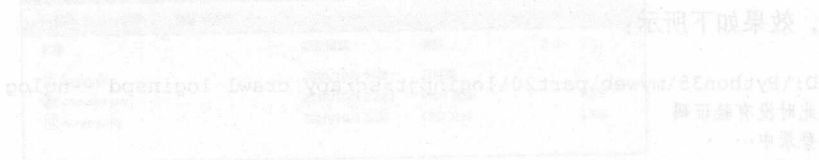


图 20-6 对应该本地文件夹下出现的验证码图片

将图片文件即为验证码图片, 查看该图片的内容, 发现如图 20-7 所示的验证码信息



图 20-7 验证码信息



韦玮

资深网络爬虫技术专家、大数据专家和软件开发工程师，从事大型软件开发与技术服务多年，现任重庆韬翔网络科技有限公司创始人兼CEO，国家专利发明人。

精通Python技术，在Python网络爬虫、Python机器学习、Python数据分析与挖掘、Python Web开发等多个领域都有丰富的实战经验。

CSDN、51CTO、天善智能等科技类社区和媒体的特邀专家和讲师，输出了大量高质量的课程和文章，深受用户喜爱。

微博：<http://weibo.com/qiansyy>

微信公众号：韦玮

**诸葛建伟** 清华大学副研究员/《Metasploit渗透测试魔鬼训练营》作者

网络爬虫是互联网上进行信息采集的通用手段，在互联网的各个专业方向上都是不可或缺的底层技术支撑。本书从爬虫基础开始，全面阐述了Python网络爬虫技术，并且包含各种经典的网络爬虫项目案例，特别是详细给出了基于Scrapy框架实现网络爬虫的最佳实践方案与流程，实战性非常强，是一本关于Python网络爬虫的优秀实战书籍，值得推荐。

**刘天斯** 腾讯高级工程师/《Python自动化运维》作者

本书详细讲解了如何基于Python从零开始构建一个成熟的网络爬虫解决方案的完整过程，以及业界主流爬虫技术的原理与实战案例，同时也引入了作者个人的经验与思考，非常有价值。本书循序渐进的内容组织结构，相信无论是新手还是老手，均能很好地阅读和吸收。

**肖力** 云技术社区创始人

网络爬虫是许多大数据分析场景的基本需求，实现爬虫程序的基本功能很简单，但是要做到自动地不间断抓取，涉及很多技术和技巧。难能可贵的是，本书将网络爬虫编程的技术和实践技巧无私地总结并分享了出来。另外，Python也是运维人的最爱，Python入门容易精通难，通过阅读本书，可以深度学习如何在一个具体场景中使用Python。

**谢佳标** 乐逗游戏高级数据分析师/《R语言游戏数据分析与挖掘》作者

Python广泛应用于网络爬虫，本书循序渐进地阐述了爬虫的理论和核心技术，以丰富的实例讲解了网络爬虫的实战应用，精心组织的代码完美地诠释了爬虫的核心要义。这本书非常值得每一位对爬虫感兴趣的读者细细研读。



# 精通Python 网络爬虫

核心技术、框架与项目实战



投稿热线: (010) 88379604  
客服热线: (010) 88379426 88361066  
购书热线: (010) 68326294 88379649 68995259

华章网站: [www.hzbook.com](http://www.hzbook.com)  
网上购书: [www.china-pub.com](http://www.china-pub.com)  
数字阅读: [www.hzmedia.com.cn](http://www.hzmedia.com.cn)

上架指导: 计算机/信息安全

ISBN 978-7-111-56208-5



9 787111 562085 >

定价: 69.00元